Application Report

Porting SimpleLink Wi-Fi Host Driver to STMicroelectronics Microcontroller

Sarah Pelosi

ABSTRACT

The SimpleLink™ Wi-Fi® family of devices provides a suite of integrated protocols for Wi-Fi and Internet connectivity to simplify the implementation of internet-enabled devices and applications.

This application report is intended as a guide to help customers port the software components needed to use a SimpleLink Wi-Fi network processor to a non-TI microcontroller, with specific references to the ST Microcontroller STM32L4xxx series. For complete details on the SimpleLink Wi-Fi host driver and porting layer, see the *Device* and *Porting the Host Driver* sections of the *SimpleLink Wi-Fi CC3x20, CC3x3x Network Processor User's Guide*.

Table of Contents

Trademarks

SimpleLink™ is a trademark of Texas Instruments.
Wi-Fi® is a registered trademark of Wi-Fi Alliance.
All trademarks are the property of their respective owners.

1 Introduction

The SimpleLink Wi-Fi device family consists of two device types: the CC32xx devices are fully integrated system-on-chip (SoC) solutions consisting of both an applications MCU and the network processor subsystem; the CC31xx devices consist only of the network processor subsystem. The CC31xx devices must be paired with a platform to run the application (MCU, MPU, or other).

The network processor subsystem firmware is ROM-based, and patches are applied by a signed binary file called a servicepack. A servicepack is required for all CC3xxx devices and must be compatible with the host driver version.

In order to use the CC31xx devices, the customer must port the host driver to their application platform. The host driver is sub-sectioned so that the customer should only modify the files in the porting layer of the host driver.

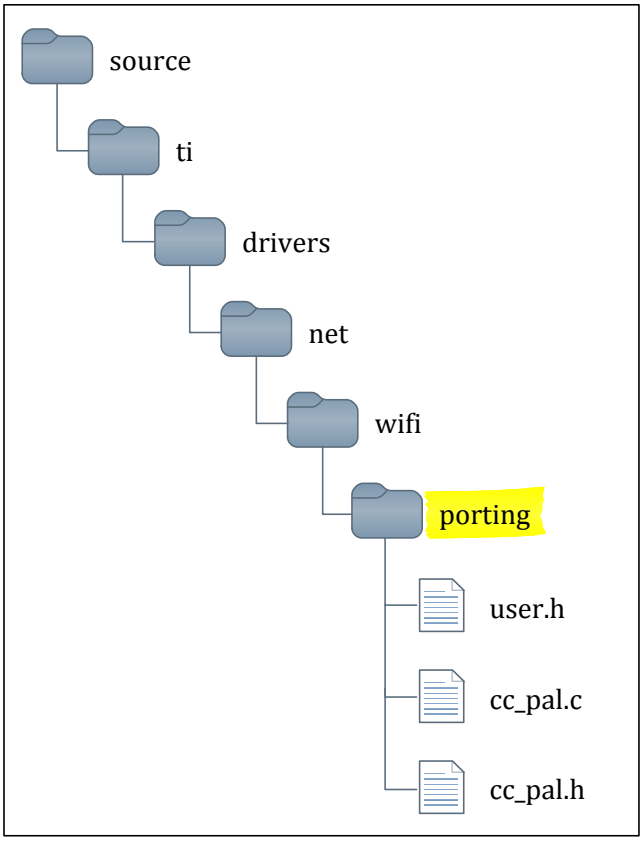
2 Porting the Host Driver

2.1 Porting Layer Files

The SimpleLink Wi-Fi host driver contains a porting layer that should be used as the adaptation layer for the host platform. In the SimpleLink Wi-Fi CC32xx SDK, this porting layer is implemented for the on-chip ARM M4 application microcontroller. The porting layer distributed in the and SimpleLink SDK Wi-Fi Plugin is implemented for other microcontrollers in the SimpleLink platform family.

This porting layer is in the folder `source/ti/drivers/net/wifi/porting`, and consists of `user.h`, `cc_pal.c`, and `cc_pal.h`. The `user.h` is a required file and acts as the interface between the SimpleLink Wi-Fi host driver APIs and the target platform. The `cc_pal` files contain the target-specific implementations used by `user.h`. The following sections will walk through each of the requirements in detail for the STMicrocontroller STM32L4xxx target platform and FreeRTOS.

The porting layer in the Wi-Fi Plugin also includes the file `SIMPLELINKWIFI.h`, which can be ignored for non-TI applications.



Since the Wi-Fi host driver is executed on the host MCU, the host driver source code (including porting files) should be added to a STM32 application. The edits to the porting files discussed in this guide use STM32L4xx HAL APIs and FreeRTOS APIs provided in the STM32Cube software packages.

2.2 Driver Enable/Disable

The CC31xx can be enabled and disabled by controlling the *nHib* signal. The host MCU controls the *nHib* signal by changing the output of a GPIO. The host can toggle the GPIO followed by a 10 millisecond delay. This delay ensures that the host driver observes the minumim hibernate time according to the timing requirements in the device-specific data sheet.

user.h:

```
#define sl_DeviceEnable()     NwpPowerOn()
#define sl_DeviceDisable()    NwpPowerOff()
```

cc_pal.h:

```
extern void NwpPowerOn(void);
extern void NwpPowerOff(void);
```

cc_pal.c:

```
void NwpPowerOn(void)
{
    HAL_GPIO_WritePin(HOST_nHIB_PORT, HOST_nHIB_PIN, 1);
}

void NwpPowerOff(void)
{
    HAL_GPIO_WritePin(HOST_nHIB_PORT, HOST_nHIB_PIN, 0);
    /* wait 10 msec */
    HAL_Delay(10);
}
```

2.3 SPI Interface

2.3.1 Hardware Setup and Configuring Clocks

The host communication interface can support SPI or UART. For this application report, the SPI interface is used.

The communication interface requires functionality for open, close, read, write, and registering an interrupt handler routine for the host IRQ signal. It is recommended that the `sl_IfOpen` open function also initializes the SPI/UART interface. The interface can be initialized outside of this function, but then it is the application's responsibility to ensure this is complete before starting the host driver.

user.h:

```
#define _SlFd_t         Fd_t
#define sl_IfOpen       spi_Open
#define sl_IfClose      spi_Close
#define sl_IfRead       spi_Read
#define sl_IfWrite      spi_Write
#define sl_IfRegIntHdlr(InterruptHdl, pValue)    NwpRegisterInterruptHandler(InterruptHdl, pValue)
```

cc_pal.h defines a suggested pin configuration for the host interface. To see the mapped pins, see Section 3.1.

cc_pal.h:

```
/* CC31xx pin configuration */
/* Definition for SPI clock resources */
#define SPIx                             SPI2
#define SPIx_CLK_ENABLE()                __SPI2_CLK_ENABLE()
#define SPIx_SCK_GPIO_CLK_ENABLE()       __GPIOB_CLK_ENABLE()
#define SPIx_MISO_GPIO_CLK_ENABLE()      __GPIOB_CLK_ENABLE()
#define SPIx_MOSI_GPIO_CLK_ENABLE()      __GPIOB_CLK_ENABLE()

#define SPIx_FORCE_RESET()               __SPI2_FORCE_RESET()
#define SPIx_RELEASE_RESET()             __SPI2_RELEASE_RESET()

/* Definition for SPIx Pins */
#define SPIx_SCK_PIN                     GPIO_PIN_13
#define SPIx_SCK_GPIO_PORT               GPIOB
#define SPIx_SCK_AF                      GPIO_AF5_SPI2
#define SPIx_MISO_PIN                    GPIO_PIN_14
#define SPIx_MISO_GPIO_PORT              GPIOB
#define SPIx_MISO_AF                     GPIO_AF5_SPI2
#define SPIx_MOSI_PIN                    GPIO_PIN_15
#define SPIx_MOSI_GPIO_PORT              GPIOB
#define SPIx_MOSI_AF                     GPIO_AF5_SPI2

/* SPI GPIO CS */
#define SPI_CS_PORT                      GPIOH
#define SPI_CS_PIN                       GPIO_PIN_13

/* Definition for SPIx's NVIC */
#define SPIx_IRQn                        SPI2_IRQn
#define SPIx_IRQHandler                  SPI2_IRQHandler

/* HOST-IRQ*/
#define HOST_IRQ_PORT                    GPIOA
#define HOST_IRQ_PIN                     GPIO_PIN_0
#define HOST_nHIB_PORT                   GPIOH
#define HOST_nHIB_PIN                    GPIO_PIN_15

extern Fd_t spi_Open(char *ifName, unsigned long flags);
extern int spi_Close(Fd_t fd);
extern int spi_Read(Fd_t fd, unsigned char *pBuff, int len);
extern int spi_Write(Fd_t fd, unsigned char *pBuff, int len);
extern int NwpRegisterInterruptHandler(P_EVENT_HANDLER InterruptHdl, void* pValue);
void EXTI0_IRQHandler(void);
```

cc_pal.c implements the SPI interface. spi_Open (defined as sl_IfOpen) sets the SPI parameters. This example uses a GPIO as a software chip select, so the SPI NSS is defined as soft. For more details about setting up the SPI clock, see Section 3.4.

cc_pal.c:

```
#define SPI_TIMEOUT_MAX 0x1000
SPI_HandleTypeDef    SpiHandle;

static void GPIO_Init(GPIO_TypeDef *GPIO_PORT, unsigned short GPIO_PIN);
static void CC31xx_InterruptEnable(void);
static void CC31xx_InterruptDisable(void);

/*******************************************************************************
* spi_Open
*******************************************************************************/
Fd_t spi_Open(char *ifName, unsigned long flags)
{
    /* Initialize the WiFi driver */
    /* Set the SPI parameters */
    SpiHandle.Instance               = SPIx;
    SpiHandle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
    SpiHandle.Init.Direction         = SPI_DIRECTION_2LINES;
    SpiHandle.Init.CLKPhase          = SPI_PHASE_1EDGE;
    SpiHandle.Init.CLKPolarity       = SPI_POLARITY_LOW;
    SpiHandle.Init.CRCCalculation    = SPI_CRCCALCULATION_DISABLED;
    SpiHandle.Init.CRCPolynomial     = 7;
    SpiHandle.Init.DataSize          = SPI_DATASIZE_8BIT;
```

```
    SpiHandle.Init.FirstBit          = SPI_FIRSTBIT_MSB;
    SpiHandle.Init.NSS               = SPI_NSS_SOFT;
    SpiHandle.Init.TIMode            = SPI_TIMODE_DISABLED;
    SpiHandle.Init.Mode              = SPI_MODE_MASTER;
    if(HAL_SPI_Init(&SpiHandle) != HAL_OK)
    {
        /* Initialization Error */
        while(1);
    }

    /* Initialize CS */
    GPIO_Init(SPI_CS_PORT, SPI_CS_PIN);
    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_SET);

    /* Enable IRQ */
    CC31xx_InterruptEnable();

    /* Initialize nHIB */
    GPIO_Init(HOST_nHIB_PORT, HOST_nHIB_PIN);
    NwpPowerOff();

    /* Wait 50 ms */
    HAL_Delay(50);
    return 0;
}

/*****************************************************************************
* spi_Close
*****************************************************************************/
int spi_Close(Fd_t fd)
{
    /* Disable Interrupt */
    CC31xx_InterruptDisable();

    /* Deinitialize SPI */
    HAL_SPI_DeInit(&SpiHandle);
    return 0;
}

/*****************************************************************************
* spi_Read
*****************************************************************************/
int spi_Read(Fd_t fd,
 unsigned char *pBuff,
 int len)
{
    /* Assert CS */
    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_RESET);
    HAL_SPI_Receive(&SpiHandle, pBuff, len, SPI_TIMEOUT_MAX);
    /* Deassert CS */
    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_SET);

    return len;
}

/*****************************************************************************
* spi_Write
*****************************************************************************/
int spi_Write(Fd_t fd,
 unsigned char *pBuff,
 int len)
{
    /* Assert CS */
    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&SpiHandle, pBuff, len, SPI_TIMEOUT_MAX);
    /* Deassert CS */
    HAL_GPIO_WritePin(SPI_CS_PORT, SPI_CS_PIN, GPIO_PIN_SET);

    return len;
}

/*****************************************************************************
* NwpRegisterInterruptHandler - set the interrupt handler function from the host driver
*****************************************************************************/
int NwpRegisterInterruptHandler(P_EVENT_HANDLER InterruptHdl,
 void* pValue)
{
    pIrqEventHandler = InterruptHdl;
```

```
    return 0;
}
```

The following code snippets are helper functions for `spi_Open` that should also go in `cc_pal.c`.

`GPIO_Init` is used to initialize the GPIOs in the host interface (*nHib* and `CS`) and their clocks.

```
/*****************************************************************************
* GPIO_Init: General GPIO initialization. Used by spi_Open
*****************************************************************************/
static void GPIO_Init(GPIO_TypeDef  *GPIO_PORT, unsigned short GPIO_PIN)
{
    GPIO_InitTypeDef   GPIO_InitStructure;

    /* Enable GPIOx clock */
    if(GPIO_PORT == GPIOA)
        __GPIOA_CLK_ENABLE();
    else if(GPIO_PORT == GPIOB)
        __GPIOB_CLK_ENABLE();
    else if(GPIO_PORT == GPIOC)
        __GPIOC_CLK_ENABLE();
    else if(GPIO_PORT == GPIOD)
        __GPIOD_CLK_ENABLE();
    else if(GPIO_PORT == GPIOE)
        __GPIOE_CLK_ENABLE();
    else if(GPIO_PORT == GPIOF)
        __GPIOF_CLK_ENABLE();
    else if(GPIO_PORT == GPIOG)
        __GPIOG_CLK_ENABLE();
    else if(GPIO_PORT == GPIOH)
        __GPIOH_CLK_ENABLE();

    /* Configure pin as input floating */
    GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStructure.Pull = GPIO_PULLUP;
    GPIO_InitStructure.Pin = GPIO_PIN;
    GPIO_InitStructure.Speed = GPIO_SPEED_FAST;

    HAL_GPIO_Init(GPIO_PORT, &GPIO_InitStructure);
}
```

`CC31xx_InterruptEnable` registers the host IRQ signal as an interrupt wake source. This host IRQ signal is required to handle asynchronous events from the network processor.

```
P_EVENT_HANDLER      pIrqEventHandler = 0;
uint32_t             irqPriority = 5;
/*****************************************************************************
* CC31xx_InterruptEnable
*****************************************************************************/
void CC31xx_InterruptEnable()
{
  /* Configure EXTI Line0 (connected to PA0 pin) in interrupt mode */
      GPIO_InitTypeDef   GPIO_InitStructure;

    /* Enable GPIOx clock */
    if(HOST_IRQ_PORT == GPIOA)
        __GPIOA_CLK_ENABLE();
    else if(HOST_IRQ_PORT == GPIOB)
        __GPIOB_CLK_ENABLE();
    else if(HOST_IRQ_PORT == GPIOC)
        __GPIOC_CLK_ENABLE();
    else if(HOST_IRQ_PORT == GPIOD)
        __GPIOD_CLK_ENABLE();
    else if(HOST_IRQ_PORT == GPIOE)
        __GPIOE_CLK_ENABLE();
    else if(HOST_IRQ_PORT == GPIOF)
        __GPIOF_CLK_ENABLE();
    else if(HOST_IRQ_PORT == GPIOG)
        __GPIOG_CLK_ENABLE();
    else if(HOST_IRQ_PORT == GPIOH)
        __GPIOH_CLK_ENABLE();

    /* Configure PA0 pin as input floating */
    GPIO_InitStructure.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStructure.Pull = GPIO_NOPULL;
```

```
    GPIO_InitStructure.Pin = HOST_IRQ_PIN;
    HAL_GPIO_Init(HOST_IRQ_PORT, &GPIO_InitStructure);

    /* Enable and set EXTI Line0 Interrupt to the lowest priority */
    HAL_NVIC_SetPriority(EXTI0_IRQn, irqPriority, 0U);
    HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}
```

`CC31xx_InterruptDisable` disables the host IRQ interrupt.

```
/*****************************************************************************
* CC31xx_InterruptDisable
*****************************************************************************/
void CC31xx_InterruptDisable()
{
    HAL_NVIC_DisableIRQ(EXTI0_IRQn);
}
```

`EXTI0_IRQHandler` is an interrupt handler for the STM32L4 EXT1 IRQ. If defined, this replaces the standard function in the HAL layer.

```
/*****************************************************************************
* @brief EXTI line detection callbacks
* @param GPIO_Pin: Specifies the pins connected EXTI line
* @retval None
*****************************************************************************/
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if((GPIO_Pin == HOST_IRQ_PIN) &&
       (NULL != pIrqEventHandler) )
    {
        pIrqEventHandler(0);
    }
}

/*****************************************************************************
* @brief This function handles External line 0 interrupt request.
* @param  None
* @retval None
*****************************************************************************/
void EXTI0_IRQHandler(void)
{
  HAL_GPIO_EXTI_IRQHandler(HOST_IRQ_PIN);
}
```

For this application, the APIs for host IRQ mask and unmask (`sl_IfMaskIntHdlr` and `sl_IfUnMaskIntHdlr`) are not used. These are typically not required for the SPI interface.

2.4 Memory Management

Memory must be allocated for the host driver's control block. A dynamic memory model can be used if `SL_MEMORY_MGMT_DYNAMIC` is defined in `user.h`. `malloc` and `free` are defined according to the FreeRTOS memory model.

user.h:

```
#include <stdlib.h>

#define sl_Malloc(Size)     pvPortMalloc(Size)
#define sl_Free(pMem)       vPortFree(pMem)
```

2.5 OS Abstraction: FreeRTOS

This application report utilizes FreeRTOS, which is included in the STM32Cube software package. This application does not use the CMSIS abstraction layer.

For guidance on a non-RTOS implementation, see the *Porting the Host Driver* sections of the *SimpleLink Wi-Fi CC3x20, CC3x3x Network Processor User's Guide*.

In `user.h`, you can see that the names given to the different `sl_Sync` and `sl_Lock` objects have a pattern. The APIs that start with "x" or "v" (such as `xSemaphoreGive`) can be directly defined as FreeRTOS APIs. The other APIs are implemented in `cc_pal`, since they need some additional parameters or error-checking to be compatible.

user.h:

```
#define SL_PLATFORM_MULTI_THREADED

#define SL_OS_RET_CODE_OK     ((int)OS_OK)

#define SL_OS_WAIT_FOREVER    ((uint32_t)OS_WAIT_FOREVER)

#define SL_OS_NO_WAIT         ((uint32_t)OS_NO_WAIT)
#define _SlTime_t             uint32_t

#define _SlSyncObj_t          SemaphoreHandle_t
#define sl_SyncObjCreate(pSyncObj,pName)    Semaphore_create(pSyncObj)
#define sl_SyncObjDelete(pSyncObj)    vSemaphoreDelete((SemaphoreHandle_t)*pSyncObj)
#define sl_SyncObjSignal(pSyncObj)    xSemaphoreGive((SemaphoreHandle_t)*pSyncObj)
#define sl_SyncObjSignalFromIRQ(pSyncObj)    Semaphore_SignalFromISR(pSyncObj)
#define sl_SyncObjWait(pSyncObj,Timeout)    Semaphore_pend_handle(pSyncObj,Timeout)
#define sl_SyncObjGetCount(pSyncObj,pValue)    Semaphore_get_count(pSyncObj, pValue);

#define _SlLockObj_t    SemaphoreHandle_t
#define sl_LockObjCreate(pLockObj, pName)    Mutex_create_handle(pLockObj)
#define sl_LockObjDelete(pLockObj)    vSemaphoreDelete((SemaphoreHandle_t)*pLockObj)
#define sl_LockObjLock(pLockObj,Timeout)    Mutex_pend_handle(pLockObj, Timeout)
#define sl_LockObjUnlock(pLockObj)    xSemaphoreGive((SemaphoreHandle_t)*pLockObj)

#define sl_GetThreadID()    xTaskGetCurrentTaskHandle()
```

cc_pal.h:

```
#include "FreeRTOS.h"
#include "semphr.h"
#include "time.h"

#define MAX_QUEUE_SIZE       (4)
#define OS_WAIT_FOREVER      (0xFFFFFFFF)
#define OS_NO_WAIT           (0)
#define OS_OK                (0)

#define Semaphore_OK         (0)
#define Semaphore_FAILURE    (-1)
#define    Semaphore_TIMEOUT    (-2)

#define Mutex_OK             (0)
#define Mutex_FAILURE        (-1)
#define    Mutex_TIMEOUT        (-2)

int Semaphore_create(SemaphoreHandle_t* pSemHandle);

int Semaphore_SignalFromISR(SemaphoreHandle_t* pSemHandle);
int Semaphore_pend_handle(SemaphoreHandle_t* pSemHandle,
 uint32_t timeout);
int Semaphore_get_count(SemaphoreHandle_t *semaphore, int *value);
int Mutex_create_handle(SemaphoreHandle_t* pMutexHandle);

int Mutex_pend_handle(SemaphoreHandle_t* pMutexHandle, uint32_t timeout);
```

cc_pal.c:

```
/*****************************************************************************
* Semaphore_create
*****************************************************************************/
int Semaphore_create(SemaphoreHandle_t* pSemHandle)
{
    //Check for NULL
    if(NULL == pSemHandle)
    {
        return Semaphore_FAILURE;
    }

    *pSemHandle = xSemaphoreCreateCounting(SEM_VALUE_MAX, 0);

    if(*pSemHandle != NULL)
    {
        return Semaphore_OK;
    }
    else
    {
        return Semaphore_FAILURE;
    }
}

/*****************************************************************************
* Semaphore_SignalFromISR
*****************************************************************************/
int Semaphore_SignalFromISR(SemaphoreHandle_t* pSemHandle)
{
    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;

    if(pdTRUE == xSemaphoreGiveFromISR((SemaphoreHandle_t)*pSemHandle, &xHigherPriorityTaskWoken))
    {
        if(xHigherPriorityTaskWoken )
        {
            taskYIELD ();
    }
        return Semaphore_OK;
    }
    else
    {

        return Semaphore_OK;
    }
}

/*****************************************************************************
* Semaphore_pend_handle
*****************************************************************************/
int Semaphore_pend_handle(SemaphoreHandle_t* pSemHandle,
 uint32_t timeout)
{
    if(SL_OS_WAIT_FOREVER == timeout)
    {
        if (pdTRUE == xSemaphoreTake((SemaphoreHandle_t)*pSemHandle, portMAX_DELAY))
        {
            return(Semaphore_OK);
    }
    else
    {
        return(Semaphore_FAILURE);
    }
    }
    else
    {
        /* usec to ticks */
    if (pdTRUE == xSemaphoreTake((SemaphoreHandle_t)*pSemHandle, \
            ( TickType_t )timeout/ClockP_getSystemTickPeriod()))
        {
            return(Semaphore_OK);
    }
    else
    {
        return(Semaphore_TIMEOUT);
    }
    }
}
```

```
/*****************************************************************************
* Semaphore_get_count
*****************************************************************************/
int Semaphore_get_count(SemaphoreHandle_t *pSemHandle, int *value)
{
    UBaseType_t count;

    count = uxSemaphoreGetCount((SemaphoreHandle_t)*pSemHandle);
    *value = (int)count;
    return 0;
}

/*****************************************************************************
* Semaphore_pend_handle
*****************************************************************************/
int Mutex_create_handle(SemaphoreHandle_t *pMutexHandle)
{
    *pMutexHandle = xSemaphoreCreateMutex();

    if(*pMutexHandle != NULL)
    {
        return Mutex_OK;
    }
    else
    {
        return Mutex_FAILURE;
    }
}

/*****************************************************************************
* Semaphore_pend_handle
*****************************************************************************/
int Mutex_pend_handle(SemaphoreHandle_t* pMutexHandle,
 uint32_t timeout)
{
    if(SL_OS_WAIT_FOREVER == timeout)
    {
        if (pdTRUE == xSemaphoreTake((SemaphoreHandle_t)*pMutexHandle, portMAX_DELAY))
    {
            return(Mutex_OK);
    }
    else
    {
        return(Mutex_FAILURE);
    }
    }
    else
    {
          /* usec to ticks */
        if (pdTRUE == xSemaphoreTake((SemaphoreHandle_t)*pMutexHandle, \
              ( TickType_t )timeout/ClockP_getSystemTickPeriod())) {
            return(Mutex_OK);
          }
          else
        {
            return(Mutex_FAILURE);
          }
    }
}
```

2.6 Timestamp Mechanism

The timestamp mechanism is required in the SimpleLink host driver. This timestamp is used to determine a timeout when the network processor does not respond.

This example uses the FreeRTOS APIs to get the tick period and the current tick count in order to calculate a timestamp.

user.h:

```
#define SL_TIMESTAMP_TICKS_IN_10_MILLISECONDS (10000 / ClockP_getSystemTickPeriod())
#define slcb_GetTimestamp TimerGetCurrentTimestamp
```

cc_pal.c:

```
uint32_t ClockP_getSystemTickPeriod()
{
    uint32_t tickPeriodUs;

    /*
     *  Tick period in microseconds. configTICK_RATE_HZ is defined in the
     *  application's FreeRTOSConfig.h, which is include by FreeRTOS.h
     */
    tickPeriodUs = 1000000 / configTICK_RATE_HZ;

    return tickPeriodUs;
}

unsigned long TimerGetCurrentTimestamp()
{
    return ((uint32_t)xTaskGetTickCount());
}
```

2.7 Asynchronous Event Handler Routines

The asynchronous event handlers are used to pass aynchronous events from the network processor to the host application. This includes fatal errors, notifications of WLAN connections and disconnections, socket events, and more. The async event handlers are defined in `user.h`, but they are implemented by the developer in the application.

The event handlers are meant to be customized for the application. There are many references for how these event handlers can be used in the Wi-Fi SDK and Plugin examples. For more details, review the UserEvents in the Host Driver API documentation in the Wi-Fi SDK or Plugin. The following code snippet defines the event handlers with the API names that are commonly used in the examples.

user.h:

```
#define slcb_DeviceFatalErrorEvtHdlr    SimpleLinkFatalErrorEventHandler

#define slcb_DeviceGeneralEvtHdlr       SimpleLinkGeneralEventHandler
#define slcb_WlanEvtHdlr                SimpleLinkWlanEventHandler
#define slcb_NetAppEvtHdlr              SimpleLinkNetAppEventHandler
#define slcb_NetAppRequestHdlr          SimpleLinkNetAppRequestEventHandler
#define slcb_NetAppRequestMemFree       SimpleLinkNetAppRequestMemFreeEventHandler
#define slcb_SockEvtHdlr                SimpleLinkSockEventHandler
```

3 Tips for Porting

3.1 Hardware Setup

The STMicroelectronics microcontroller STM32L4xx is available for development as a STM32L4R9 Discovery Kit (32L4R9IDISCOVERY). This can be blue-wired to any of the CC31xx BoosterPack Evaluation Kits, such as CC3120BOOST or BOOSTXL-CC3135.

Signal	32L4R9IDISCOVERY	CC3120BOOST / BOOSTXL-CC3135
nHIB	CN10-1 (PH15)	P1.5
IRQ	CN17-5 (PA0)	P2.19
SPI Clock	CN10-6 (PB13)	P1.7
SPI MISO	CN10-5 (PB14)	P2.14
SPI MOSI	CN10-4 (PB15)	P2.15
SPI CS	CN10-2 (PH13)	P2.18
3V3	CN16-4 (3V3)	P1.1
GND	CN16-6 (GND)	P2.20

3.2 Servicepack

The SimpleLink Wi-Fi network processor is ROM-based and, therefore, requires that patches be applied from a servicepack file. This signed servicepack file must be loaded to the serial flash.

For more information on how to flash a CC31xx BoosterPack, see the *UniFlash ImageCreator User's Guide*.

3.3 Starting the Wi-Fi Driver in the Application Code

There are a few things that must be done in the host MCU application before it can successfully start the CC31xx device. In a RTOS environment, the host driver requires that the application create a spawn contect called `sl_Task`. This thread serves the asynchronous requests from the network processor, and it should be a high enough priority in the system to handle networking events. For platforms without an OS, the application must call the `sl_Task` function repeatedly from its main loop.

Example of a main function:

```
/* Customize sl_Task priority and stack size for your application */
#define sl_PRIORITY     ( tskIDLE_PRIORITY + 2UL )
#define sl_STACK_SIZE    768  /* FreeRTOS stack size is specified in words */

int main(void) {
    BaseType_t xStatus;

    /* STM32L4xx HAL library initialization */
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();

    /* Start sl_Task thread */
    xStatus = xTaskCreate( (TaskFunction_t)sl_Task, "sl_Task", sl_STACK_SIZE, NULL, sl_PRIORITY,
NULL);
    if (xStatus != pdPASS)
    {
        printf("\n Unable to create sl_Task\r\n");
    }

    /* Other application threads can be started here too */

    /* Start the RTOS scheduler */
    vTaskStartScheduler();
}
```

After creating the `sl_Task` thread and starting the RTOS scheduler, the application can call `sl_Start(0,0,0)` to start the network processor and begin using the Wi-Fi APIs.

3.4 Configuring Clocks on the STM32L4

The STMicroelectronics microcontroller requires you to configure the system and peripheral clocks. In a STM32L4 demo, the system clock is typically configured in `SystemClock_Config`, which is called in `main`. The peripheral clocks are typically configured within the `stm32l4xx_hal_msp.c` file. In this guide, the GPIO clocks are configured (for *nHib*, *CS*, and so forth) in `cc_pal.c`. The SPI peripheral clock can be configured in `stm32l4xx_hal_msp.c` by including the `cc_pal.h` header file. `HAL_SPI_MspInit` and `HAL_SPI_MspDeInit` are standard functions from the STM32L4 HAL library.

```
#include "cc_pal.h"

/**
  * @brief SPI MSP Initialization
  *        This function configures the hardware resources used in this example:
  *           - Peripheral's clock enable
  *           - Peripheral's GPIO Configuration
  * @param hspi: SPI handle pointer
  * @retval None
  */
void HAL_SPI_MspInit(SPI_HandleTypeDef *hspi)
{
    GPIO_InitTypeDef  GPIO_InitStruct;

    if(hspi->Instance == SPIx)
    {

        /* Enable GPIO TX/RX clock */
        SPIx_SCK_GPIO_CLK_ENABLE();
        SPIx_MISO_GPIO_CLK_ENABLE();
        SPIx_MOSI_GPIO_CLK_ENABLE();
        /* Enable SPI clock */
        SPIx_CLK_ENABLE();


        /* SPI SCK GPIO pin configuration  */
        GPIO_InitStruct.Pin       = SPIx_SCK_PIN;
        GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull      = GPIO_PULLDOWN;
        GPIO_InitStruct.Speed     = GPIO_SPEED_FAST;
        GPIO_InitStruct.Alternate = SPIx_SCK_AF;
        HAL_GPIO_Init(SPIx_SCK_GPIO_PORT, &GPIO_InitStruct);

        /* SPI MISO GPIO pin configuration  */
        GPIO_InitStruct.Pin = SPIx_MISO_PIN;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        HAL_GPIO_Init(SPIx_MISO_GPIO_PORT, &GPIO_InitStruct);

        /* SPI MOSI GPIO pin configuration  */
        GPIO_InitStruct.Pin = SPIx_MOSI_PIN;
        HAL_GPIO_Init(SPIx_MOSI_GPIO_PORT, &GPIO_InitStruct);


      /* NVIC for SPI */
     HAL_NVIC_SetPriority(SPIx_IRQn, 1, 0);
      HAL_NVIC_EnableIRQ(SPIx_IRQn);
    }
}

/**
  * @brief SPI MSP De-Initialization
  *        This function frees the hardware resources used in this example:
  *          - Disable the Peripheral's clock
  *          - Revert GPIO configuration to its default state
  * @param hspi: SPI handle pointer
  * @retval None
  */
void HAL_SPI_MspDeInit(SPI_HandleTypeDef *hspi)
{
    if(hspi->Instance == SPIx)
    {
        /* Reset peripherals */
        SPIx_FORCE_RESET();
        SPIx_RELEASE_RESET();

        /* Configure SPI SCK as alternate function  */
```

```
        HAL_GPIO_DeInit(SPIx_SCK_GPIO_PORT, SPIx_SCK_PIN);
        /* Configure SPI MISO as alternate function  */
        HAL_GPIO_DeInit(SPIx_MISO_GPIO_PORT, SPIx_MISO_PIN);
        /* Configure SPI MOSI as alternate function  */
        HAL_GPIO_DeInit(SPIx_MOSI_GPIO_PORT, SPIx_MOSI_PIN);

      /* Disable the NVIC for SPI */
      HAL_NVIC_DisableIRQ(SPIx_IRQn);
    }
}
```

If the application is using FreeRTOS without the CMSIS abstraction layer, the `SysTick_Handle` in `stm32l4xx_it.c` should also be updated:

```
/* Comment out #include "cmsis_os.h" */

/**
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @retval None
  */
void SysTick_Handler(void)
{

    HAL_IncTick();

    if (xTaskGetSchedulerState() != taskSCHEDULER_NOT_STARTED)
    {
        xPortSysTickHandler();
    }
}
```

3.5 Terminal I/O Printing

To enable debug printing (`printf`) in IAR Embedded Workbench with the 32L4R9IDISCOVERY, go to the project options, open General Options, and select the Library Configuration. Set the interface implementation as **Semihosted** and stdout/stderr as **Via semihosting**.

Start a debug session, then open **View -> Terminal I/O**.

3.6 Location of the Host Driver and Porting Files

The host driver assumes that the host driver files are located in the `source/ti/drivers/net/wifi` folder of the Wi-Fi SDK or Plugin.

If you want to copy the host driver and porting files elsewhere, the following changes should be made:

1. Add the new directory location of the host driver files to the project's include paths
2. Inside `simplelink.h`, change `#include <ti/drivers/net/wifi/porting/user.h>` to `#include "porting/user.h"`

3.7 Updating to the Latest Host Driver Version

The CC3120/CC313x devices share the same host driver with the CC3220/CC323x devices, and the networking capabilities are similar.

The CC32xx SDK is released on a quarterly basis, so it will often have a more up-to-date host driver and servicepack than the Wi-Fi Plugin. To use the latest host driver version, you need to install the CC32xx SDK and grab the entire `source/ti/drivers/net/wifi` folder. Inside the `wifi` folder, you can replace the files in the `porting` folder according to this document. You must also flash the latest servicepack for your device from the `tools/cc32xx_tools/servicepack-*` folder.

4 References

- Texas Instruments: *SimpleLink Wi-Fi CC3x20, CC3x3x Network Processor User's Guide*
- Host Driver API documentation
- Texas Instruments: *UniFlash ImageCreator User's Guide*

5 License Information

This License Information will take precedence over anything in this Application Report to the contrary.

The unmodified files user.h,cc_pal.c and cc_pal.h are governed by the licenses included in the headers within the files themselves.

Portions of the suggested modifications to user.h, cc_pal.c, and cc_ppal.h related to FreeRTOS set forth in this application report are governed by the following license:

Copyright © 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Portions of all other suggested modifications to user.h, cc_pal.c, and cc_ppal.h set forth in this application report are governed by the following license:

Copyright 2017 STMicroelectronics

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Re-distributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Re-distributions 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.
3. Neither the name of the copyright holder 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.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265