Approach Note

Version 1.0

May 10, 2013

CC3000 - Host Driver - Multithread Support

Contents

Introduction

This document briefly describes the approach that was followed for adding multithreaded support to the existing host-driver. It also explains in brief, the application that uses these interfaces/APIs for exploiting the multithreaded host-driver capabilities.

This document doesn't intend to describe the architecture of host-driver or other sister components.

Design Overview

- A 'Select-Thread' and 'Lock-Objects' (seven in total) are introduced to ensure synchronization and critical-section integrity in the new CC3000 host-driver with multithread support.
    - 'Lock-Objects'
        - Main-Lock: This lock prevents the API functions from being called simultaneously by multiple threads – It ensures that only one thread has SPI access at a given time
        - Accept-Lock: It blocks the caller until 'Select-Thread' gets a response for the accept request
        - Socket-Locks (LS1 to LS4): CC3000 allows four sockets to coexist. All these sockets will have their own lock objects to enable easy notification to the caller.
        - Sleep-Lock: Controls the interworking of various user threads with 'Select-Thread'
    - 'Select-Thread'
        - It runs as long as WLAN is enabled and monitors multiple sockets, waiting until one or more of the descriptors become "ready" for some class of I/O operation. A descriptor is considered ready if it is possible to perform the corresponding I/O operation without blocking.
        - It also synchronizes the execution of various user-threads running in the system.

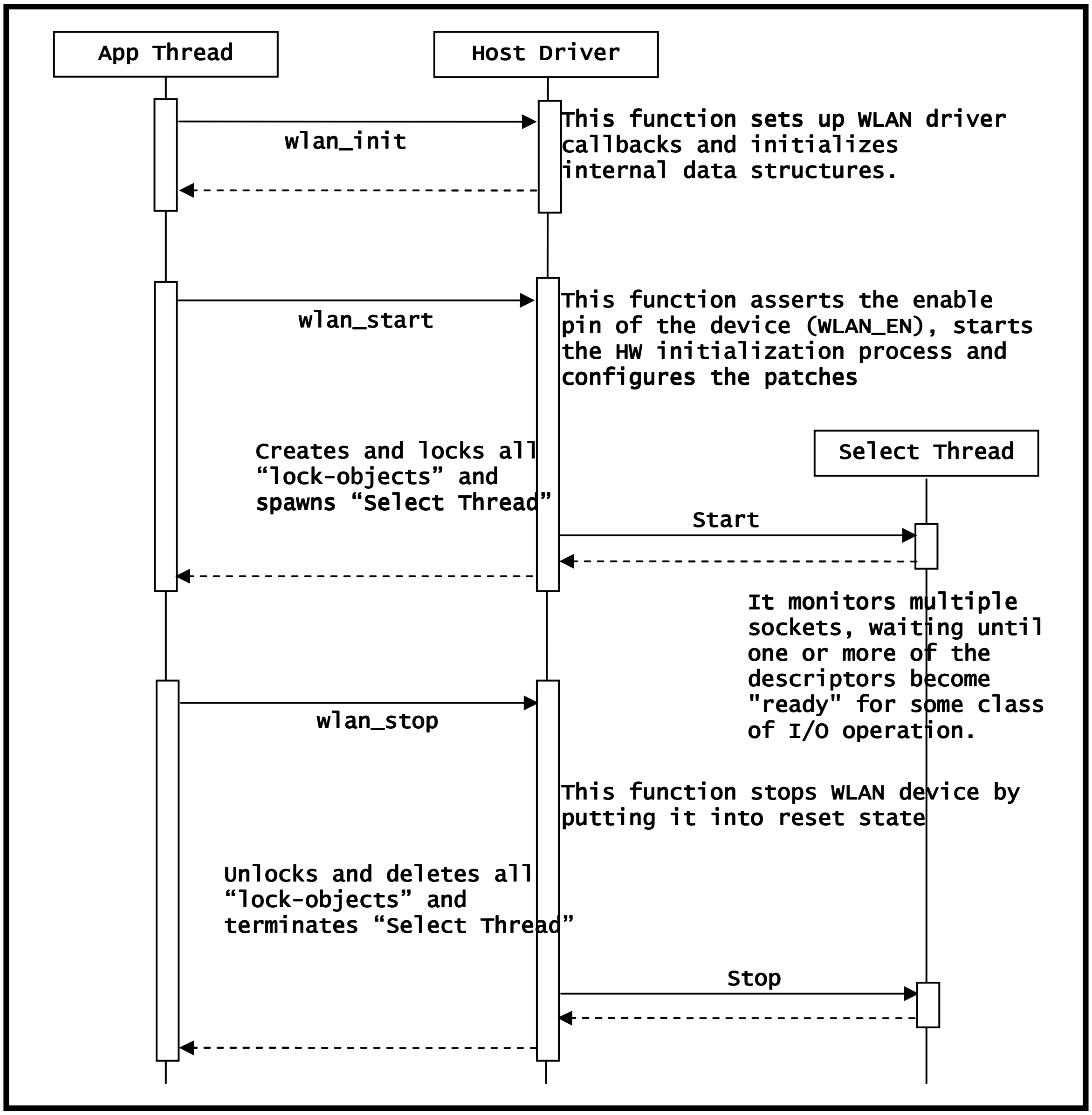
- APIs in multithreaded host-driver implementation will use 'Main-Lock' to ensure that every function is fully executed by the thread that got control.

- 'Accept' function will be asynchronous in the new design – The polling on 'Accept' will be done by 'Select-Thread' and the later is responsible to wake up the 'Accept' caller thread.

- All long operation functions, such as "recv", "recvfrom" or "accept", will use two 'Lock-Objects' to synchronize
    - First is a 'Socket-Lock' (LS1-LS4) or 'Accept-Lock'. The caller will wait here until the 'Select-Thread' releases the lock.
    - Second is the 'Main-Lock' which prevents multiple threads from using the API simultaneously.

Pseudo code and sequence flow

Initialization and termination sequence



This function sets up WLAN driver callbacks and initializes internal data structures.

This function asserts the enable pin of the device (WLAN_EN), starts the HW initialization process and configures the patches

Creates and locks all “lock-objects” and spawns “Select Thread”

It monitors multiple sockets, waiting until one or more of the descriptors become "ready" for some class of I/O operation.

This function stops WLAN device by putting it into reset state

Unlocks and deletes all “lock-objects” and terminates “Select Thread”

Diagram 1: Initialization and termination sequence

Long operation sequence (“recv”/“recvfrom”)

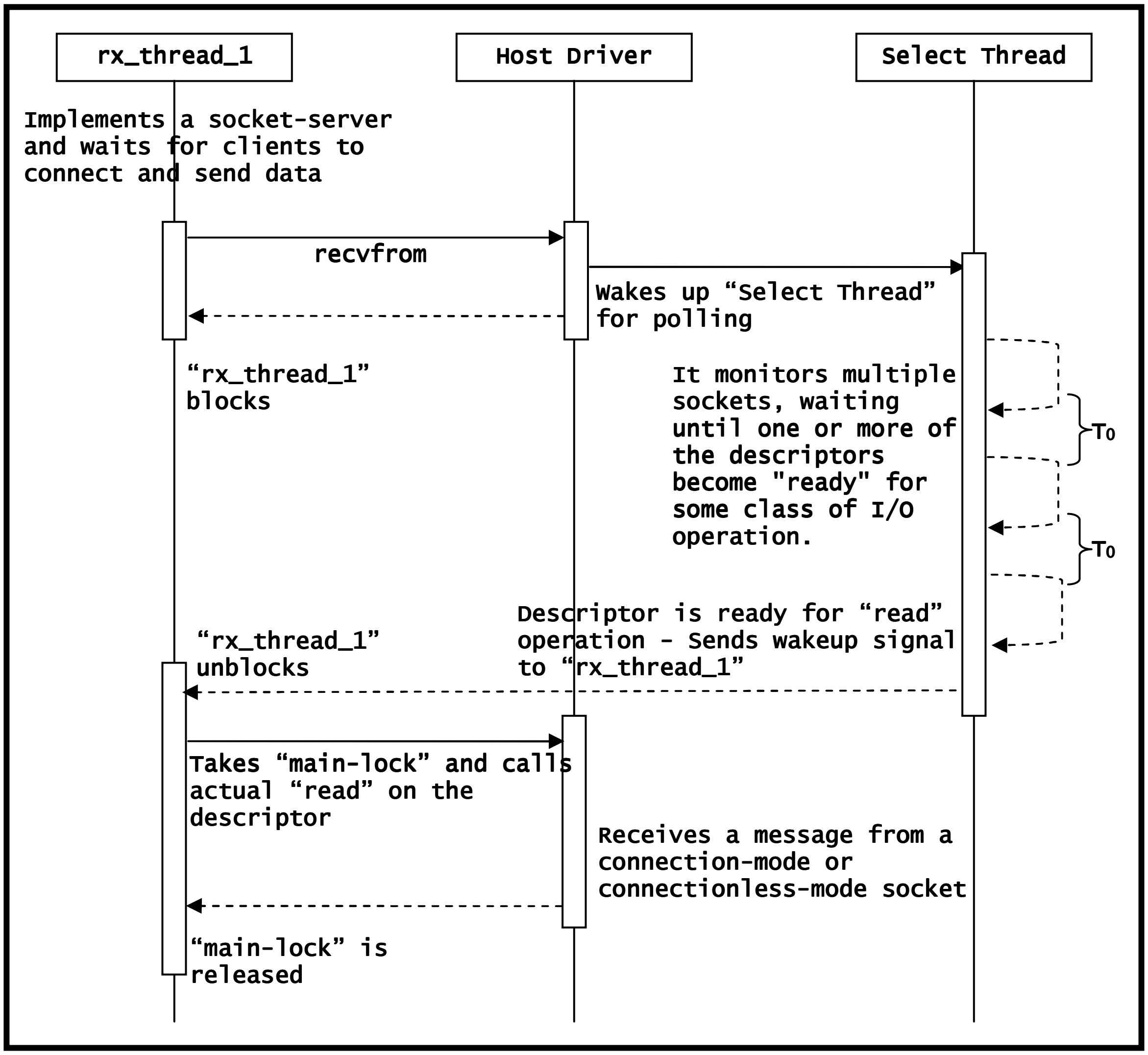

rx_thread_1
Host Driver
Select Thread
Implements a socket-server and waits for clients to connect and send data
recvfrom
Wakes up “Select Thread” for polling
“rx_thread_1” blocks
It monitors multiple sockets, waiting until one or more of the descriptors become "ready" for some class of I/O operation.
To
To
Descriptor is ready for “read” operation - Sends wakeup signal to “rx_thread_1”
“rx_thread_1” unblocks
Takes “main-lock” and calls actual “read” on the descriptor
Receives a message from a connection-mode or connectionless-mode socket
“main-lock” is released

Diagram 2: Long operation sequence

Pseudo code - Select Thread

```
Input: None

Loop: Till WLAN is ON
      Pend: (Sleep-Lock)                #waiting on a semaphore
      Post: (Sleep-Lock)                #releasing a semaphore
      Initialize: (FD-List)             #max is 4
      If: Read/Write Descriptor? SET
                                 ”select” every $T_0$ milliseconds
           Loop:
                Post: (Socket-Lock)     #releasing a semaphore
                                #LS1 to LS4 whichever is ready
           End Loop: Max num of sockets #four in case of CC3000
      If: Poll on “Accept”? TRUE
          Lock: (Main-Lock)           #enter critical section
              Accept                  #asynchronously
          Unlock: (Main-Lock)         #leave critical section
End Loop:                              #run until WLAN is ON
```

Snippet 1: Select-Thread pseudo code

Pseudo code – Long operation (“recvfrom”)

```
Input: [In] Socket Handle
       [Out] Buffer
       [In] Length of Buffer
       [In] Flags
       [In] Source Address
       [In] Source Address Length

Post: (Sleep-Lock)                 #releasing a semaphore
                                   #wake-up select thread
Pend: (Socket-Lock)                #LS1 to LS4
Pend: (Sleep-Lock)                 #for suspending Select-Thread
Lock: (Main-Lock)                  #enter critical section
    Receive Data                   #Non-blocking now
Unlock: (Main-Lock)                #leave critical section
```

Snippet 2: Long operation pseudo code

Software Structure

Applications, both single-threaded and multithreaded, can be developed with the same API set.

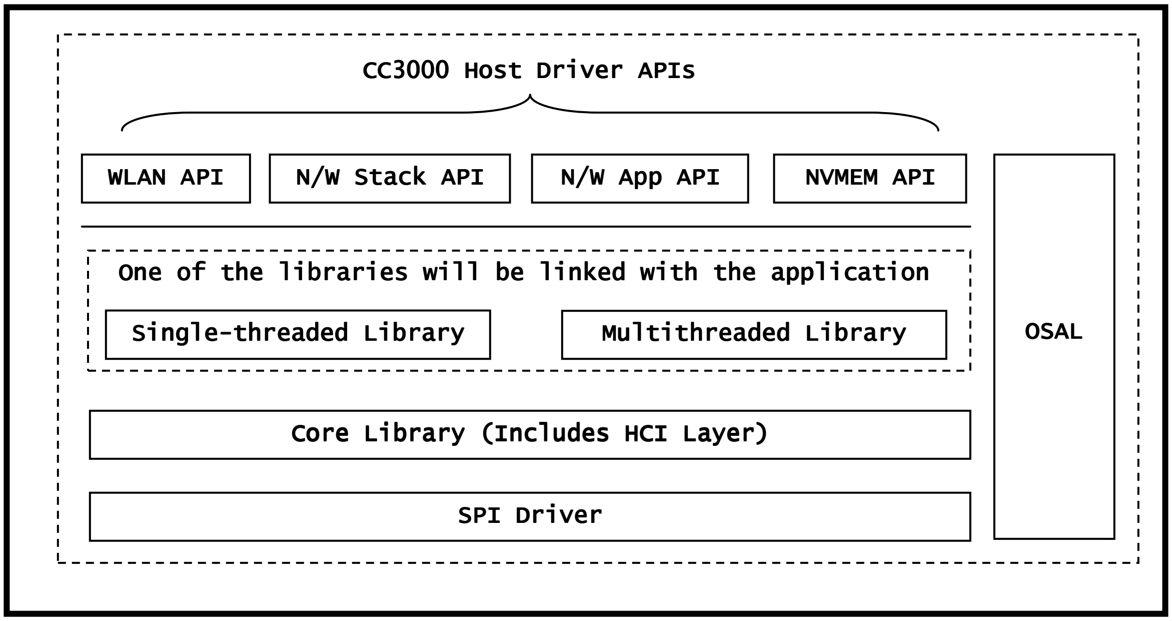


Diagram 3: Software Structure

Application developers should include either single-threaded or multithreaded library, depending on the type of application they intend to develop. Detailed guidelines are provided in “CC3000 Host Driver Multithread Support User Guide”.

Application Definition

A sample application is provided with the package to showcase the multithreaded host-driver capabilities. It’s a client-server application that transmits and receives data in parallel. It connects to a Wi-Fi network and spawns two tasks when launched, a socket-server (“rx_thread_1” & “rx_thread_2”) and a socket-client (“tx_thread”), one waiting for data from an ‘iperf-client’ and the other sending data to an ‘iperf-server’ respectively.

Sequence Flow

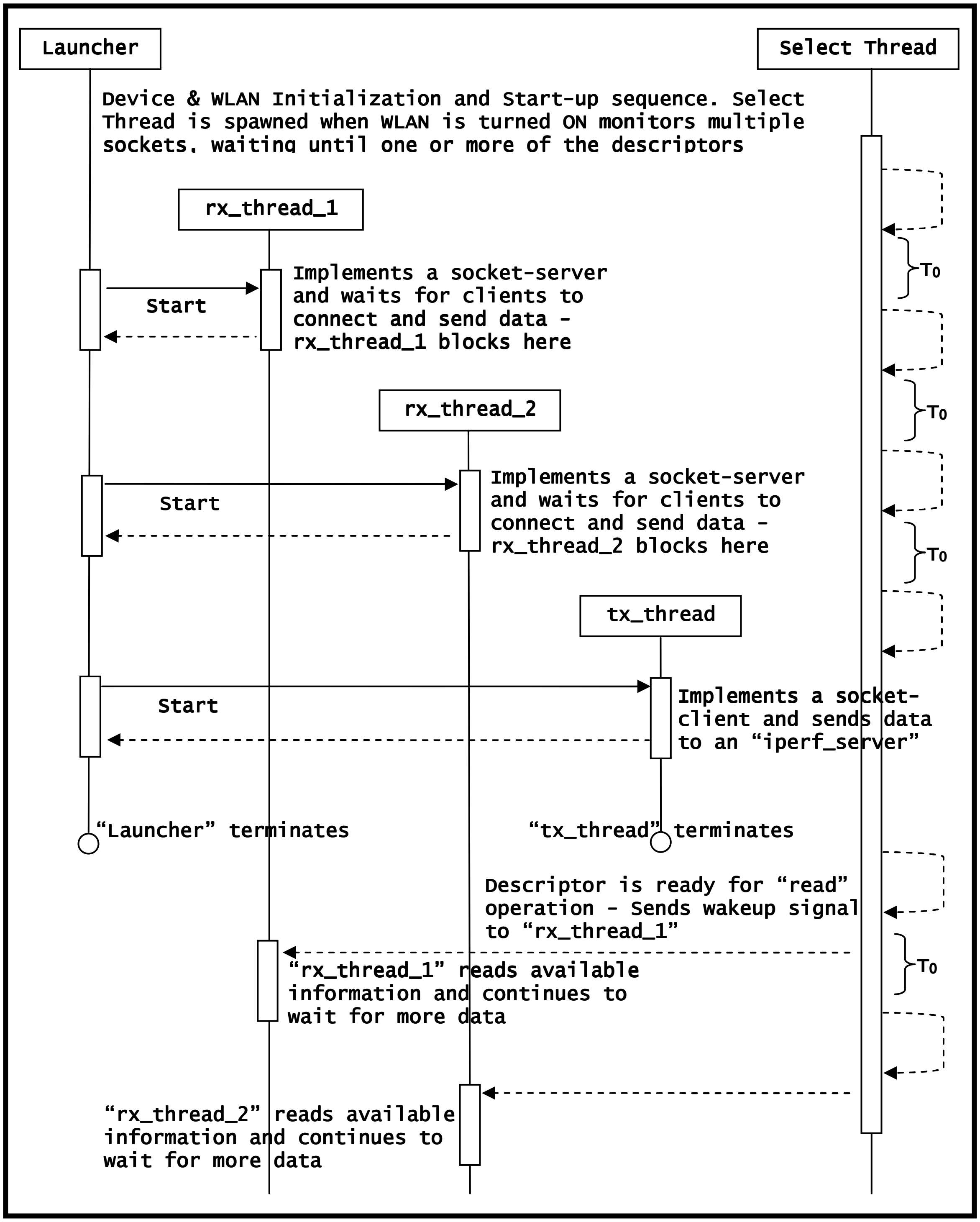

Launcher
Select Thread
Device & WLAN Initialization and Start-up sequence. Select Thread is spawned when WLAN is turned ON monitors multiple sockets, waiting until one or more of the descriptors
rx_thread_1
Start
Implements a socket-server and waits for clients to connect and send data - rx_thread_1 blocks here
To
rx_thread_2
Start
Implements a socket-server and waits for clients to connect and send data - rx_thread_2 blocks here
To
To
tx_thread
Start
Implements a socket-client and sends data to an "iperf_server"
"Launcher" terminates
"tx_thread" terminates
Descriptor is ready for "read" operation - Sends wakeup signal to "rx_thread_1"
To
"rx_thread_1" reads available information and continues to wait for more data
"rx_thread_2" reads available information and continues to wait for more data

Diagram 2: Long operation sequence