LWIP port for tirtos

Final project for Advanced Computer System – Prof. Sivan Toledo

Git:
https://github.com/adamelha/lwip_tirtos

By:

Adam Elhakham

201390085

About LWIP

LWIP (Light Weight IP) is an open source TCP/IP stack designed for embedded systems. LWIP features implementations for the link layer, IP layer (over multiple network interfaces), transport layer, and application layer. It includes implementations for many protocols.

Check out the git repo:

https://git.savannah.gnu.org/git/lwip/lwip-contrib.git

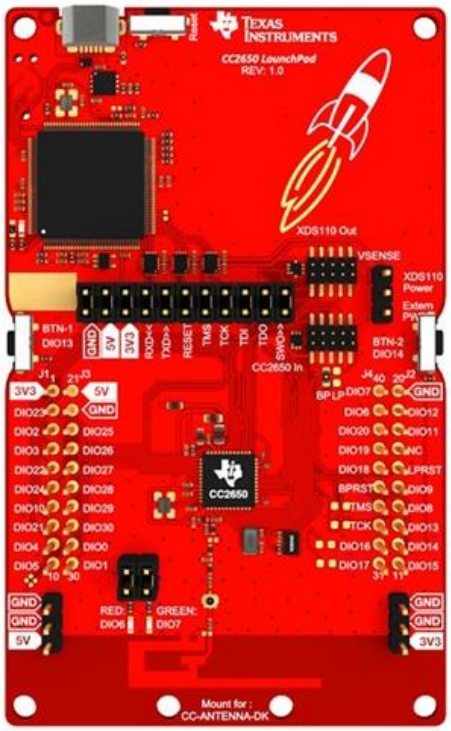
Note that there are two main directories:

- Lwip – which contains the lwip core (IP layer implementation, TCP, etc..), library APIs (tcp-ip APIs, sockets, etc..) and apps (http server, DNS service, etc..).
- Lwip-contrib – which contain more example apps (socket examples, usage of TCP/UDP APIs, etc..). It also contains ports for UNIX and Windows systems, and several embedded real time operating systems.

Unfortunately, lwip-contrib does not contain a port to the Ti-rtos operating system that we used during the course. This is where this project comes in handy.

I created a port for Ti-rtos. This port can be used in order to use the LWIP library on the Texas Instruments CC2650 that we used during the course:

The CC2650



The device uses an ARM Cortex M3 main processor and up to 48-MHZ clock speed. For more information about the CC2650:

http://www.ti.com/product/CC2650/datasheet

The plain CC2650 does not support wifi or Ethernet connection. Therefore, the purpose of this project is to enable the device to use LWIP with the SLIP protocol as the network interface.

The SLIP protocol:

> The SLIP (Serial Line Internet protocol) is a simple encapsulation of the IP protocol. SLIP requires a serial port connection to connect to the "outside world". SLIP is not used on PCs anymore but still used with microcontrollers due to its small overhead. SLIP is documented in RFC 1055.

In this project I use SLIP as the network interface for the IP stack. Fortunately, LWIP already implements a SLIP network interface.

The Project:

This project consist of two main parts:

1. The Ti-rtos port.
2. An Example UDP server app that we can use to obtain data collected by the CC2650 using sockets.

All the code is in the git repo I created:

https://github.com/adamelha/lwip_tirtos

The Ti-rtos Port

We will now dive in to the port itself that will enable us to use the LWIP library (and also create the example app).

All the code for the port is located in lwip-contrib/ports/tirtos in according to the LWIP convension.

The port files:

- **Include/arch/cc.h:**

  This header file contains compiler defines such as format qualifiers, Endianity, but also debug and assertion macros.

  Note: I created a flag that enables a system flush of the debug buffer after every debug print so that no debug information will be lost. This is not recommended but I left it as an option.

```
/*
 * If LWIP_DEBUG_WITH_FLUSH is defined to 1, every debug print will immediately be displayed in ROV.
 * This is not recommended since it slows execution significantly. Although can Help debugging a bit.
 *
 * If LWIP_DEBUG_WITH_FLUSH is 0, user must use System_flush() to flush debug buffer.
 */
#define LWIP_DEBUG_WITH_FLUSH   0
```

- **Include/arch/sys_arch.h:**

  This is a Header extends lwip sys.h. This contains the interface of synchronizing system APIs such as counting semaphores, mutexes (binary semaphores), mailboxes (All supported in ti-rtos). As well as threads (I used ti-rtos tasks to implement the threading interface).

  Helpful defines for the user in this file:

```
/* Maximum message buffer for mailboxes */
#define MAX_MAILBOX_MESSAGE_SIZE    1024

/* Maximum size of a name for a thread */
#define MAX_THREAD_NAME_SIZE        100

/* Defining this will treat every clock tick as a millisecond.
 * This is instead of calling bios API BIOS_getCpuFreq which is not thread safe and will require additional locking */
#define DEFAULT_1MS_CLOCK_TICK
```

- **lwipopts.h:**

  This is the lwip configuration file. Several changes had to be made for the memory pools to fit the constraints of the CC2650.

  For SLIP we defined the following:

```
418 /*
419    ----------------------------------------
420    ---------- SLIP options ----------
421    ----------------------------------------
422 */
423
424 #define SLIPIF_THREAD_NAME              "slipif_loop"
425
426 #define SLIPIF_THREAD_STACKSIZE          1000
427
428 #define SLIPIF_THREAD_PRIO               4
429
430 #define SLIP_USE_RX_THREAD               1
431
432 #define USE_SLIPIF                       1
433
```

Helpful debug options:

```
434 /*
435    ----------------------------------------
436    ---------- Debugging options ----------
437    ----------------------------------------
438 */
439
440 #define TAPIF_DEBUG       LWIP_DBG_OFF
441 #define TUNIF_DEBUG       LWIP_DBG_OFF
442 #define UNIXIF_DEBUG      LWIP_DBG_OFF
443 #define DELIF_DEBUG       LWIP_DBG_OFF
444 #define SIO_FIFO_DEBUG    LWIP_DBG_OFF
445 #define TCPDUMP_DEBUG     LWIP_DBG_OFF
446 #define API_LIB_DEBUG     LWIP_DBG_OFF
447 #define API_MSG_DEBUG     LWIP_DBG_OFF
448 #define TCPIP_DEBUG       LWIP_DBG_OFF
449 #define NETIF_DEBUG       LWIP_DBG_ON
450 #define SOCKETS_DEBUG     LWIP_DBG_OFF
451 #define DEMO_DEBUG        LWIP_DBG_OFF
452 #define IP_DEBUG          LWIP_DBG_ON
453 #define IP_REASS_DEBUG    LWIP_DBG_OFF
454 #define RAW_DEBUG         LWIP_DBG_OFF
455 #define ICMP_DEBUG        LWIP_DBG_OFF
456 #define UDP_DEBUG         LWIP_DBG_ON
457 #define TCP_DEBUG         LWIP_DBG_OFF
458 #define TCP_INPUT_DEBUG   LWIP_DBG_OFF
459 #define TCP_OUTPUT_DEBUG  LWIP_DBG_ON
460 #define TCP_RTO_DEBUG     LWIP_DBG_OFF
461 #define TCP_CWND_DEBUG    LWIP_DBG_OFF
462 #define TCP_WND_DEBUG     LWIP_DBG_OFF
463 #define TCP_FR_DEBUG      LWIP_DBG_OFF
464 #define TCP_QLEN_DEBUG    LWIP_DBG_OFF
465 #define TCP_RST_DEBUG     LWIP_DBG_OFF
466 #define SLIP_DEBUG        LWIP_DBG_ON
```

- **sys_arch.c:**

  This is the source file that implements all the interfaces in sys_arch.h.

- **sio.c:**

  This source file implements lwip/sio.h and is required for LWIP's implementation of the SLIP network interface.

  This file implements all the SERIAL communication. Implemented using APIs from the ti-rtos ti/drivers/UART.h.

  Use the following define to configure the desired baud rate for serial communication:

```
#define UART_BAUD_RATE  115200
```

This concludes the implementation of the Ti-rtos port for LWIP.

The Example App

In addition to the Ti-rtos port, also provided is an example app.

The app provides a way to make queries from a remote computer, using standard Python UDP sockets, to the CC2650 which will get the device to check the room temperature or its battery situation (from the Ti-rtos batmon module).

The App consists of three parts:

1. **Client side – ports/tirtos/SlipPyhton/client.py**

   This python script is used by the remote PC. It opens a UDP socket and sends the device 3 different commands:
   - 'TEMP' – to get the room temperature of the remote CC2650.
   - 'BAT' – to get the voltage supplied to the remote CC2630.
   - 'NOT A COMMAND'– to demonstrate the device response from an invalid command.

2. **Server side – ports/tirtos/SlipPython/server.py**

   This python script is used by the PC that is connected to the CC2650 and is the "middleman" between the CC2650 and the outside world.

   It runs two separate threads:
   - A thread that uses Scapy to sniff IP packets, encode them to SLIP and write the encapsulated SLIP packets to the serial line (where the CC2650 is connected).
   - A thread that polls the serial line for SLIP IP packets from the CC2650. When a SLIP packet is read, it decodes it and sends the IP packet using Scapy send().

3. **CC2650 App**

   This is an app that uses LWIP's UDP APIs and runs a UDP server with a SLIP network interface that listens on port 1000 on the IP of the PC running server.py.

The App files:

- lwip_udp_server.c

  Contains main(). Does board initializations, initializes app task and starts bios.

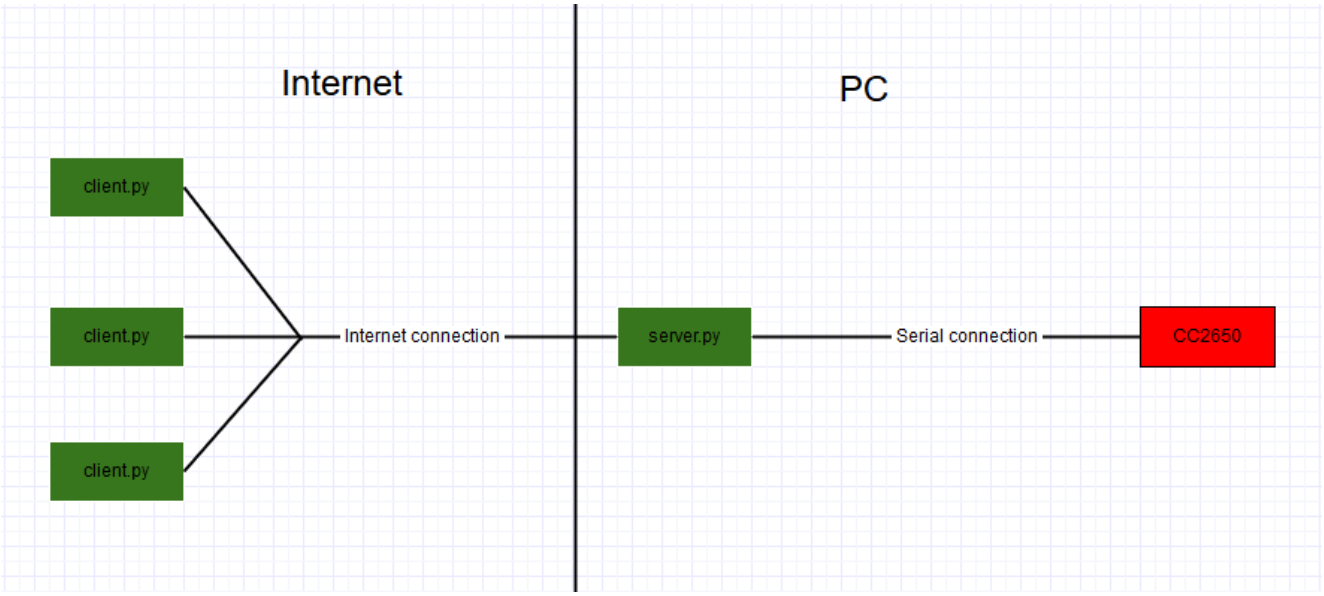
- ports/tirtos/slip_task .c & .h

  Creates the app task and calls apps start function.

- ports/tirtos/main.c

  Makes initializations of LWIP and SLIP network interface which uses a thread to poll the serial line for SLIP packets. When a Complete SLIP packet arrives, it is decoded and fed to the IP layer. If UDP packet to the configured IP and port arrives, a callback functions is called. The callback extracts the data from the packet, and based on this data, it decides to either determine the room temperature, or the battery voltage, and send the response, or send a BAD API response.

App Diagram:

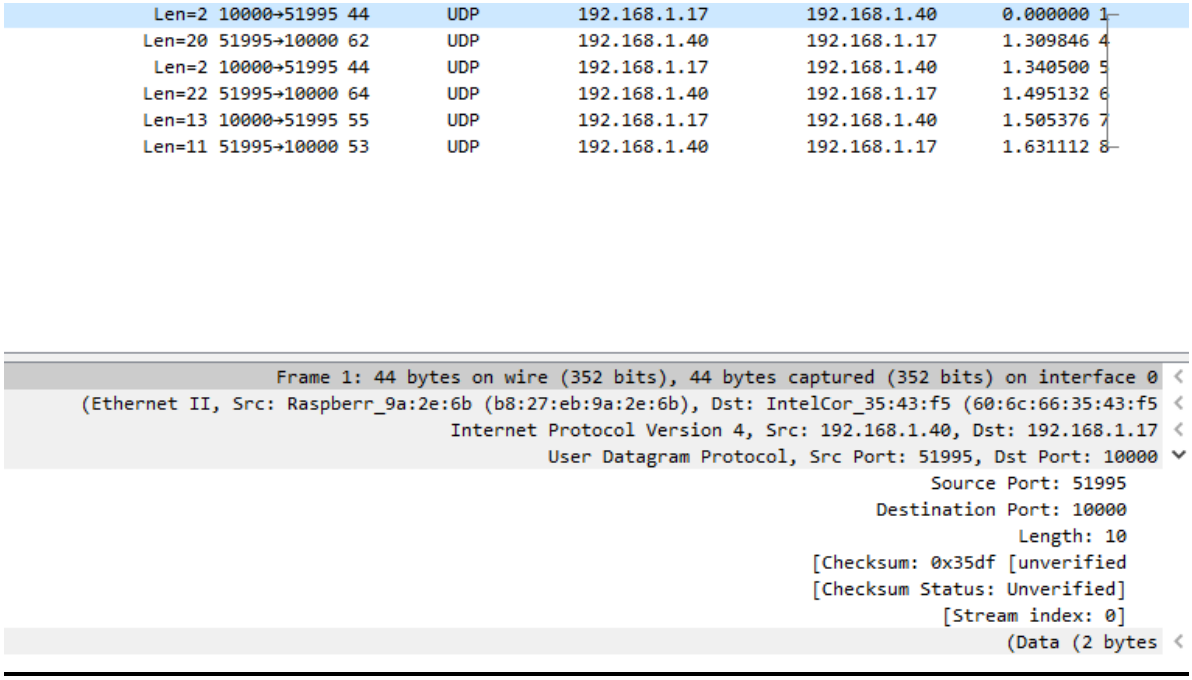


Client.py logs:

```
C:\dev>python client.py
Sent command TEMP : Gets device room temperature in celsius
data from ('192.168.1.17', 10000): Device Response: 29
Sent command BAT : Gets device battery in volts
data from ('192.168.1.17', 10000): Device Response: 3.3203
Sent command NOT A COMMAND :  This API does not exist
data from ('192.168.1.17', 10000): Device Response: API Error!

C:\dev>C:\dev
```

Wireshark snippet showing the UDP ping pong between client.py and the CC2650:



Challenges

This project provided many challenges.

- The most time consuming was configuring the project to fit a small memory foot print and not overflow from the defined stack.
  Stack overflows caused things to work part of the time and sometimes just caused the app to hang. Especially with the serial communication.

- Another major challenge was learning how LWIP works in order to even start building an LWIP app and debugging it. The LWIP library is a very large piece of code and even though it is pretty well documented, it was difficult to dive into it.

- Synchronizing between different tasks on the CC2650 was also challenging.

- The Python apps also provided difficulty especially since I couldn't get Scapy to work properly on Windows. It is takes it very long to load.