lwIP 1.3.0 Library (v1.00.b)

UG 650 April 15, 2009

Summary

This document describes the Embedded Development Kit (EDK) port of the open source lightweight IP (lwIP) TCP/IP stack. The lwIP provides an easy way to add TCP/IP-based networking capability to an embedded system.

The `lwip130_v1_00_b` library provides adapters for the `xps_ethernetlite` and `xps_ll_temac` Xilinx® Ethernet MAC cores, and is based on the lwIP stack version 1.3.0. This document describes how to use the `lwip130_v1_00_b` library to add networking capability to embedded software. It contains the following sections:

- "Overview"
- "Features"
- "Additional Resources"
- "Using lwIP"
- "Setting up the Hardware System"
- "Setting up the Software System"
- "lwIP Performance"
- "Known Issues and Restrictions"
- "Migrating from lwip_v3_00_a to lwip130_v1_00_b"
- "API Examples"
- "Software APIs"
- "Appendix A: lwIP Source Documentation"

Overview

The lwIP is an open source TCP/IP protocol suite available under the BSD license. The lwIP is a standalone stack; there are no operating systems dependencies, although it can be used along with operating systems. The lwIP provides two APIs for use by applications:

- RAW API: Provides access to the core lwIP stack.
- Socket API: Provides a BSD sockets style interface to the stack.

The `lwip130_v1_00_b` is an EDK library that is built on the open source lwIP library version 1.3.0. The `lwip130_v1_00_b` library provides adapters for the Ethernetlite (`xps_ethernetlite`) and the TEMAC (`xps_ll_temac`) Xilinx EMAC cores. The library can run on MicroBlaze™, PowerPC® 405, or PowerPC 440 processors.

Features

The lwIP provides support for the following protocols:

- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- User Datagram Protocol (UDP)
- TCP (Transmission Control Protocol (TCP)
- Address Resolution Protocol (ARP)
- Dynamic Host Configuration Protocol (DHCP)

Additional Resources

- lwIP wiki: http://lwip.wiki.com
- Xilinx lwIP designs and application examples: http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf
- lwIP examples using RAW and Socket APIs: http://savannah.nongnu.org/projects/lwip/
- *Multi-Port Memory Controller (MPMC) Data Sheet:* available in the following directory of your software installation: `EDK\hw\XilinxProcessorIPLib\pcores\mpmc_v*_00_a`

Using lwIP

The following sections detail the hardware and software steps for using lwIP for networking in an EDK system. The key steps are:

1. Creating a hardware system containing the processor, ethernet core, and a timer. The timer and ethernet interrupts must be connected to the processor using an interrupt controller.
2. Configuring the `lwip130_v1_00_b` library to be a part of the software platform. For lwIP socket API, the Xilkernel library is a pre-requisite.

Setting up the Hardware System

This section describes the hardware configurations supported by lwIP. The key components of the hardware system include:

- Processor: either a PowerPC (405 or 440) processor or a MicroBlaze processor.
- EMAC: lwIP supports `xps_ethernetlite` and `xps_ll_temac` EMAC cores
- Timer: to maintain TCP timers, lwIP requires that certain functions are called at periodic intervals by the application. An application can do this by registering an interrupt handler with a timer.
- DMA: the `xps_ll_temac` core can be configured with an optional Soft Direct Memory Access (SDMA) engine

The following figure shows a system architecture in which the system is using an `xps_ethernetlite` core.

The system has a processor connected to a Multi-Port Memory Controller (MPMC) with the other required peripherals (timer and ethernetlite) on the PLB v4.6 bus. Interrupts from both the timer and the ethernetlite are required, so interrupts are connected to the interrupt controller.

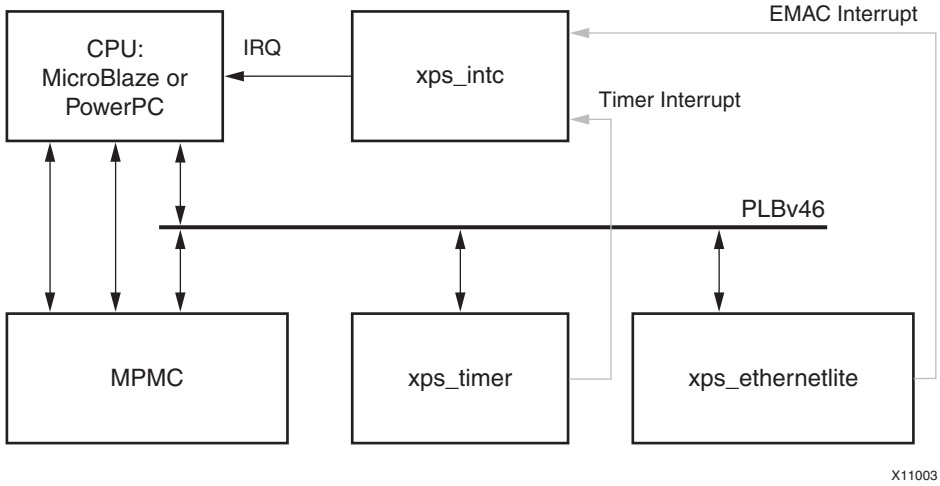

Figure 1: **System Architecture using xps_ethernetlite Core**

When using TEMAC, the system architecture changes depending on whether DMA is required. If DMA is required, a fourth port (of type SDMA), which provides direct connection between the TEMAC (`xps_ll_temac`) and the memory controller (MPMC), is added to the memory controller. The following figure shows this system architecture.

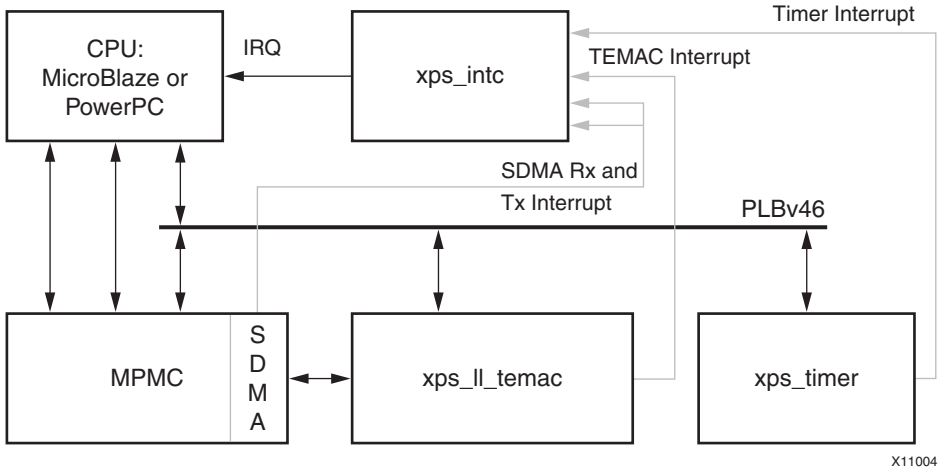

Figure 2: **System Architecture using xps_ll_temac Core (with DMA)**

Note: There are four interrupts that are necessary in this case: a timer interrupt, a TEMAC interrupt, and the SDMA RX and TX interrupts. The SDMA interrupts are from the Multi-Port Memory Controller (MPMC) SDMA Personality Interface Module (PIM). Refer to the *Multi-Port Memory Controller (MPMC) Data Sheet* for more information.

If the TEMAC is used without DMA, a FIFO (`xps_ll_fifo`) is used to interface to the TEMAC. The system architecture in this case is shown in the following figure.

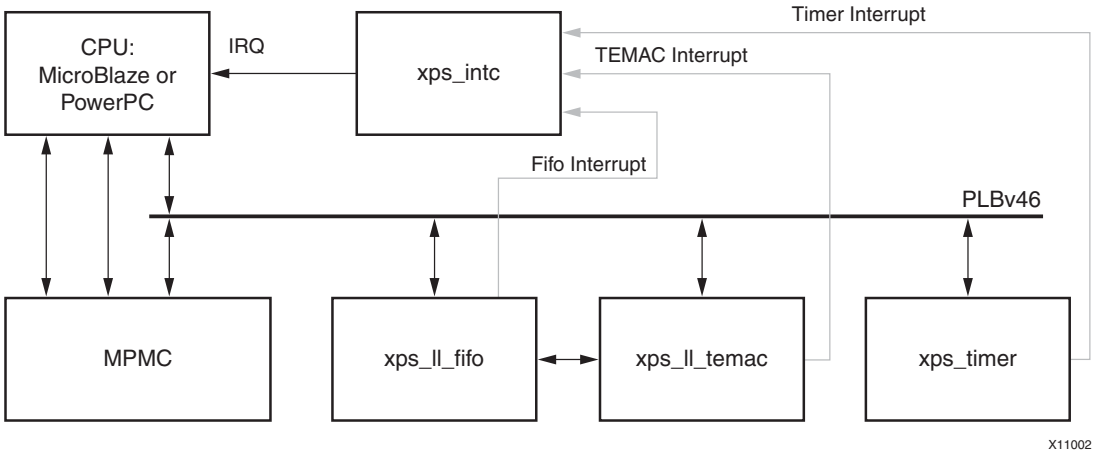

Figure 3: **System Architecture using TEMAC with xps_ll_fifo (without DMA)**

Setting up the Software System

To use lwIP in a software application, you must first compile the library as part of your application software platform. To set up the lwIP library in XPS:

1. Open the **Software Platform Settings** dialog box.
2. Enable lwIP in the **Library/OS Settings** tab. (For Socket API, Xilkernel must be the OS, configured with semaphores, mutexes, and yield functionality available).
3. Select **Generate Libraries and BSPs** to regenerate the library.
4. Link the application with the **`-l lwip4`** linker flag. (For socket API, add **`-l xilkernel`**.)

Configuring lwIP Options

The lwIP provides configurable parameters. The values for these parameters can be changed using the **Software Platform Settings** dialog box. There are two major categories of configurable options:

- Xilinx Adapter to lwIP options: These control the settings used by Xilinx adapters for the `xps_ethernetlite` and `xps_ll_temac` cores.
- Base lwIP options: These options are part of lwIP library itself, and include parameters for TCP, UDP, IP and other protocols supported by lwIP.

The following sections describe the available lwIP configurable options.

Customizing lwIP API Mode

The `lwip130_v1_00_b` library supports both raw API and socket API:

- The raw API is customized for high performance and lower memory overhead. The limitation of raw API is that it is callback-based, and consequently does not provide portability to other TCP stacks.
- The socket API provides a BSD socket-style interface and is very portable; however, this mode is inefficient both in performance and memory requirements.

The `lwip130_v1_00_b` library also provides the ability to set the priority on TCP/IP and other lwIP application threads. The following table provides lwIP library API modes.

Table 1: **API Mode Options and Descriptions**

Attribute/Options	Type	Default	Description
api_mode {RAW_API | SOCKET_API}	enum	RAW_API	The lwIP library mode of operation
socket_mode_thread_prio *integer*	integer	1	Priority of lwIP TCP/IP thread and all lwIP application threads. This setting applies only when Xilkernel is used in priority mode. It is recommended that all threads using lwIP run at the same priority level.

Xilkernel scheduling policy when using Socket API

The lwIP library in socket mode requires the use of the Xilkernel, which provides two policies for thread scheduling: round-robin and priority based:

There are no special requirements when round-robin scheduling policy is used because all threads receive the same time quanta.

With priority scheduling, care must be taken to ensure that lwIP threads are not starved. lwIP internally launches all threads at the priority level specified in `socket_mode_thread_prio`. In addition, application threads must launch `xemacif_input_thread`. The priorities of both `xemacif_input_thread`, and the lwIP internal threads (`socket_mode_thread_prio`) must be high enough in relation to the other application threads so that they are not starved.

Configuring Xilinx Adapter Options

The Xilinx adapters for `xps_ethernetlite` and `xps_ll_temac` EMAC cores are configurable.

Ethernetlite Adapter Options

The following table provides the configuration parameters for the `xps_ethernetlite` adapter.

Table 2: **xps_ethernetlite Adapter Options**

Attribute	Type	Default	Description
`sw_rx_fifo_size`	integer	8192	Software Buffer Size in bytes of the receive data between EMAC and processor
`sw_tx_fifo_size`	integer	8192	Software Buffer Size in bytes of the transmit data between processor and EMAC

TEMAC Adapter Options

The following table provides the configuration parameters for xps_ll_temac adapter.

Table 3: **xps_ll_temac Adapter**

Attribute	Type	Default	Description
phy_link_speed {CONFIG_LINKSPEED10| CONFIG_LINKSPEED100| CONFIG_LINKSPEED1000| CONFIG_LINKSPEED_AUTODETECT}	integer	CONFIG_LINKSPEED_ AUTODETECT	Link speed as auto-negotiated by the PHY. lwIP configures the TEMAC for this speed setting. This setting must be correct for the TEMAC to transmit or receive packets. ***Note:*** The setting, CONFIG_LINKSPEED_AUTODETECT, attempts to detect the correct linkspeed by reading the PHY registers; however, this is PHY dependent, and has been tested with the Marvell PHYs present on Xilinx development boards. For other PHYs, the correct speed should be chosen.
n_tx_descriptors	integer	32	Number of TX buffer descriptors used in SDMA mode
n_rx_descriptors	integer	32	Number of RX buffer descriptors used in SDMA mode
n_tx_coalesce	integer	1	TX interrupt coalescing setting for the TEMAC
n_rx_coalesce	integer	1	RX interrupt coalescing setting for the TEMAC
tcp_tx_csum_offload	integer	1	TX enable checksum offload
tcp_rx_csum_offload	integer	1	RX enable checksum offload

Configuring Memory Options

lwIP stack provides different kinds of memories. The configurable memory options are provided as a separate category. Default values work well unless application tuning is required. The various memory parameter options are provided in the following table:

Table 4: **Memory Configuration Parameter Options**

Attribute	Type	Default	Description
mem_size	int	8192	Size of the heap memory in bytes. Set this value high if application sends out large data.
mem_num_pbuf	int	16	Number of memp struct pbufs. Set this value high if application sends lot of data out of ROM or static memory.
mem_num_udp_pcb	int	5	Number of active UDP protocol control blocks. One per active UDP connection.
mem_num_tcp_pcb	int	5	Number of active TCP protocol control blocks. One per active TCP connections.
mem_num_tcp_pcb_listen	int	5	Number of listening TCP connections.
mem_num_tcp_seg	int	255	Number of simultaneously queued TCP segments.
mem_num_sys_timeout	int	3	Number of simultaneously active time-outs.

Configuring Socket Memory Options

Sockets API mode has memory options. The configurable socket memory options are provided as a separate category. Default values work well unless application tuning is required. The following table provides the parameters for the socket memory options.

Table 5: **Socket Memory Options Configuration Parameters**

Attribute	Type	Default	Description
memp_num_netbuf	int	5	Number of struct netbufs. This translates to one per socket.
memp_num_netconn	int	5	Number of struct netconns. This translates to one per socket.
memp_num_api_msg	int	8	Number of struct api_msg. Used for communication between TCP/IP stack and application.
memp_num_tcpip_msg	int	8	Number of struct tcpip_msg. Used for sequential API communication and incoming packets.

Note: Because Sockets Mode support uses Xilkernel services, the number of semaphores chosen in the Xilkernel configuration must take the value set for the memp_num_netbuf parameter into account.

Configuring Packet Buffer (Pbuf) Memory Options

Packet buffers (Pbufs) carry packets across various layers of the TCP/IP stack. The following are the pbuf memory options provided by the lwIP stack. Default values work well unless application tuning is required. The following table provides the parameters for the Pbuf memory option:

Table 6: **Pbuf Memory Options Configuration Parameters**

Attribute	Type	Defaults	Description
pbuf_pool_size	int	512	Number of buffers in pbuf pool.
pbuf_pool_bufsize	int	1536	Size in bytes of each pbuf in pbuf pool.

Configuring ARP Options

The following table provides the parameters for the ARP options. Default values work well unless application tuning is required.

Table 7: **ARP Options Configuration Parameters**

Attribute	Type	Default	Description
arp_table_size	int	10	Number of active hardware addresses, IP address pairs cached.
arp_queueing	int	1	When enabled, (default (1)), outgoing packets are queued during hardware address resolution.
arp_queue_first	int	0	When enabled, first packet queued is not overwritten by later packets. The default (0), disabled, is recommended.
etharp_always_insert	int	0	When set to 1, cache entries are updated or added for every ARP traffic. This option is recommended for routers. When set to 0, only existing cache entries are updated. Entries are added when lwIP is sending to them. Recommended for embedded devices.

Configuring IP Options

The following table provides the IP parameter options. Default values work well unless application tuning is required.

Table 8: **IP Configuration Parameter Options**

Attribute	Type	Default	Description
ip_forward	int	0	Set to 1 for enabling ability to forward IP packets across network interfaces. If running lwIP on a single network interface, set o 0.
ip_reassembly	int	1	Reassemble incoming fragmented IP packets.
ip_frag	int	1	Fragment outgoing IP packets if their size exceeds MTU.
ip_options	int	0	When set to 1, IP options are allowed (but not parsed). When set to 0, all packets with IP options are dropped.

Configuring ICMP Options

The following table provides the parameter for ICMP protocol option. Default values work well unless application tuning is required.

Table 9: **ICMP Configuration Parameter Option**

Attribute	Type	Default	Description
icmp_ttl	int	255	ICMP TTL value.

Configuring UDP Options

The following table provides UDP protocol options. Default values work well unless application tuning is required.

Table 10: **UDP Configuration Parameter Options**

Attribute	Type	Defaults	Description
lwip_udp	bool	true	Specify if UDP is required.
udp_ttl	int	255	UDP TTL value.

Configuring TCP Options

The following table provides the TCP protocol options. Default values work well unless application tuning is required.

Table 11: **TCP Options Configuration Parameters**

Attribute	Type	Defaults	Description
lwip_tcp	bool	true	Require TCP.
tcp_ttl	int	255	TCP TTL value.
tcp_wnd	int	16384	TCP Window size in bytes.
tcp_maxrtx	int	12	TCP Maximum retransmission value.
tcp_synmaxrtx	int	4	TCP Maximum SYN retransmission value.
tcp_queue_ooseq	int	1	Accept TCP queue segments out of order. Set to 0 if your device is low on memory.
tcp_mss	int	1476	TCP Maximum segment size.
tcp_snd_buf	int	32768	TCP sender buffer space in bytes.

Configuring Debug Options

lwIP stack has debug information. The debug mode can be turned on to dump the debug messages onto STDOUT. The following option, when set to true, prints the debug messages.

Table 12: **Debug Options Configuration Parameters**

Attribute	Type	Default	Description
lwip_debug	bool	false	Turn on lwIP Debug

Configuring the Stats Option

lwIP stack has been written to collect statistics, such as the number of connections used; amount of memory used; and number of semaphores used, for the application. The library provides the `stats_display()` API to dump out the statistics relevant to the context in which the call is used. The stats option can be turned on to enable the statistics information to be collected and displayed when the `stats_display` API is called from user code. Use the following option to enable collecting the stats information for the application.

Table 13: **Statistics Options Configuration Parameters**

Attribute	Type	Default	Description
`lwip_stats`	int	0	Turn on lwIP Statistics

lwIP Performance

This section provides a brief overview of the expected performance when using lwIP with Xilinx Ethernet MACs.

The following table provides the maximum TCP throughput achievable by FPGA, CPU, EMAC, and system frequency in RAW and Socket modes. Applications requiring high performance should use the RAW API.

Table 14: **Library Performance**

FPGA	CPU	EMAC	System Frequency	Max TCP Throughput	
				RAW Mode	Socket Mode
Virtex	PPC405	`xps_ll_temac`	100 MHz	140 Mbps	40 Mbps
Virtex	Microblaze	`xps_ll_temac`	125 MHz	100 Mpbs	30 Mbps
Spartan	Microblaze	`xps_ll_temac`	66 MHz	35 Mpbs	10 Mbps
Spartan	Microblaze	`xps_ethernetlite`	66 MHz	15 Mbps	7 Mbps

Known Issues and Restrictions

The lwip130_v1_00_b library does not support more than one TEMAC within a single xps_ll_temac instance. For example, the `lwip130_v1_00_b` library does not support the TEMAC enabled by setting `C_TEMAC1_ENABLED` = 1 in `xps_ll_temac`.

Migrating from lwip_v3_00_a to lwip130_v1_00_b

Applications written to work with lwip_v3_00_a must make the following changes to work with the `lwip130_v1_00_b` library:

- The API for function `sys_thread_new` has changed from lwIP 1.2.0 to lwIP 1.3.0. Use the new API as follows:

  ```
  sys_thread_t sys_thread_new(char *name, void (*thread)(void *arg), void
  *arg, int stacksize, int prio);
  ```

- Configure Xilkernel to include yield functionality.
- UDP RAW mode callback functions receive a pointer to the IP address of the sender as one of the parameters. Do not pass this parameter back to any other UDP function as an argument. Instead, make a copy and pass a pointer to the copy.