# Using Ethernet With MicroBlaze

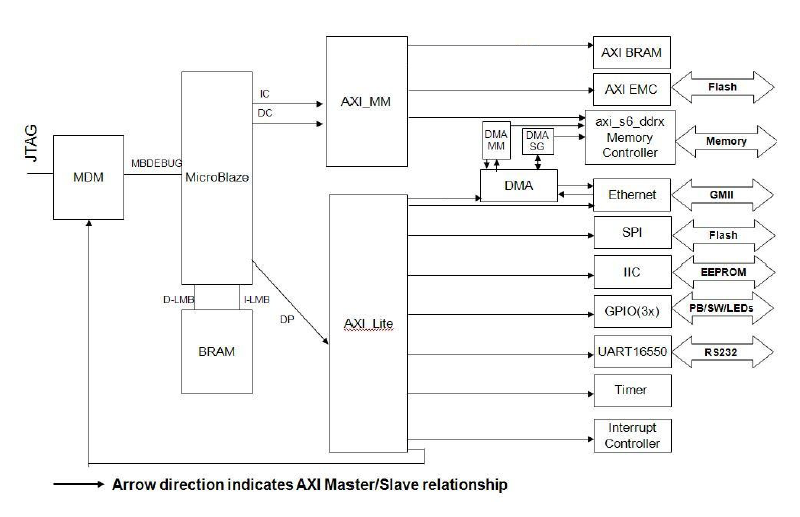
Before diving into the design part for Ethernet usage with Microblaze, some information must be given. This information will be useful for software part. Hardware design has some constraints to run the software and these constraints will be discussed under these headers.

# What is lwIP(leight weight IP)?

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. LwIP support IP, ICMP, UDP, TCP, ARP, DHCP, IGMP protocols.

LwIP libraries provides adapters for the AXI\_Ethernetlite, the TEMAC (which will be used in this design), and the Gigabit Ethernet Controller. The library can run on MicroBlaze and ARM Cortex -A9 processors. To work with lwIP some hardware and software are required. LwIP functions are used with real time operating systems. Old resources say available library for MicroBlaze is xil\_kernel for Microblaze, but now FreeRTOS can be used and also is recommended. Hardware requirements are as follows:

* **Processor**: Either a MicroBlaze or a Cortex-A9 processor. The Cortex-A9 processor applies to Zynq systems.
* **MAC**: LwIP supports axi\_ethernetlite, axi\_ethernet, and Gigabit Ethernet controller and MAC (GigE) cores.
* **Timer**: to maintain TCP timers, lwIP raw API based applications require 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**: For axi\_ethernet based systems, the axi\_ethernet cores can be configured with a soft DMA engine or a fifo interface. For GigE-based Zynq systems, there is a built-in DMA and so no extra configuration is needed. Same applies to axi\_ethernetlite based systems, which have their built-in buffer management provisions.



## FreeRTOS

Before using lwIP functions concept of RTOS must be understood. In this section basics of FreeRTOS is mentioned.

* Following libraries must be included to use FreeRTOS in projects.

|  |
| --- |
| **#include** "FreeRTOS.h"  **#include** "task.h"  **#include** "queue.h"  **#include** "timers.h" |

* To create a new task xTaskCreate function is used as follows

|  |
| --- |
| **xTaskCreate**( nameOfTheTask, textNameForTheTask, StackSizeToBeAllocatedForTask, TaskParameter, TastPriority, TaskHandlePointer); |

* Tasks must be in infinite loop. The task below prints “Hello” to screen in every second. pdMS\_TO\_TICKS converts the time value in millisecond to tick for timer. TickType\_t is a type definition for timer ticks and it is 32 bits or 16 bits unsigned integer according to the processor.

|  |
| --- |
| **static** **void** **prvTxTask**( **void** \*pvParameters )  {  **const** TickType\_t x1second = pdMS\_TO\_TICKS( 1000 );  **for**( ;; )  {  /\* Delay for 1 second. \*/  vTaskDelay( x1second );  xil\_printf("Hello\n");  }  } |

* Queues are used to provide communication between tasks. They are created by using xQueueCreate() function. This function takes two parameters. First parameter is the space in the queue and the second is the size of data to hold. Following example created a queue called xQueue, size of integer with 2 spaces.

|  |
| --- |
| **static** QueueHandle\_t xQueue = NULL;  xQueue = xQueueCreate( 2,**sizeof**(**int**) ); |

* To read data in the queue xQueueReceive() function is used with 3 parameters which are accordingly the queue being read, pointer to hold read data, time to wait the queue. Following example waits for the data to be read from xQueue to ReceivedData 1000 ticks.

|  |
| --- |
| xQueueReceive(xQueue,Recdstring, 1000); |

* To write data to the queue xQueueReceive() function is used with 3 parameters which are accordingly the queue being written, pointer of the data being written, and block time.

|  |
| --- |
| Char SendData = ‘A’;  xQueueSend(xQueue, SendData, 0UL ); |

* Timer is created using xTimerCreate() function 5 parameter which are accordingly timer name to use to assist in debugging, period in ticks, auto reload option true or false, timer ID and finally the name of callback function.

|  |
| --- |
| **static** TimerHandle\_t xTimer = NULL;  xTimer = xTimerCreate( (**const** **char** \*) "Timer",x10seconds,  pdFALSE, (**void** \*) TIMER\_ID, vTimerCallback); |

* Timer is initialized with xTimerStart( xTimerHandler, Blocktime ); function.
* Task are initialized with vTaskStartScheduler(); function.
* Tasks are closed with vTaskDelete( xTaskHandle ); function.

## Designing the Hardware

## Designing the Software