Cómo conectar una Zynq por Ethernet

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/06/23/como-conectar-una-zynq-por-ethernet/

Blog: https://soceame.wordpress.com/

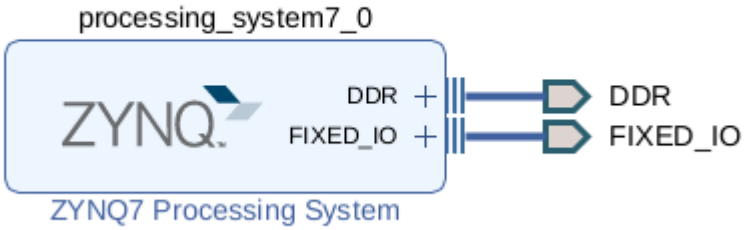
GitHub: https://github.com/DRubioG

Fecha última modificación: 22/02/2025

Los SoCs de Xilinx tienen la gran ventaja de que llevan embebido un hard-core, este hard-core permite ejecutar aplicaciones dentro que en una FPGA serían bastante complejas. Una de ellas es la conexión por Ethernet. Bien, pues este problema Xilinx lo ha resuelto utilizando la librería **LWIP (lightweight IP).** Ahora te explico cómo usarla.

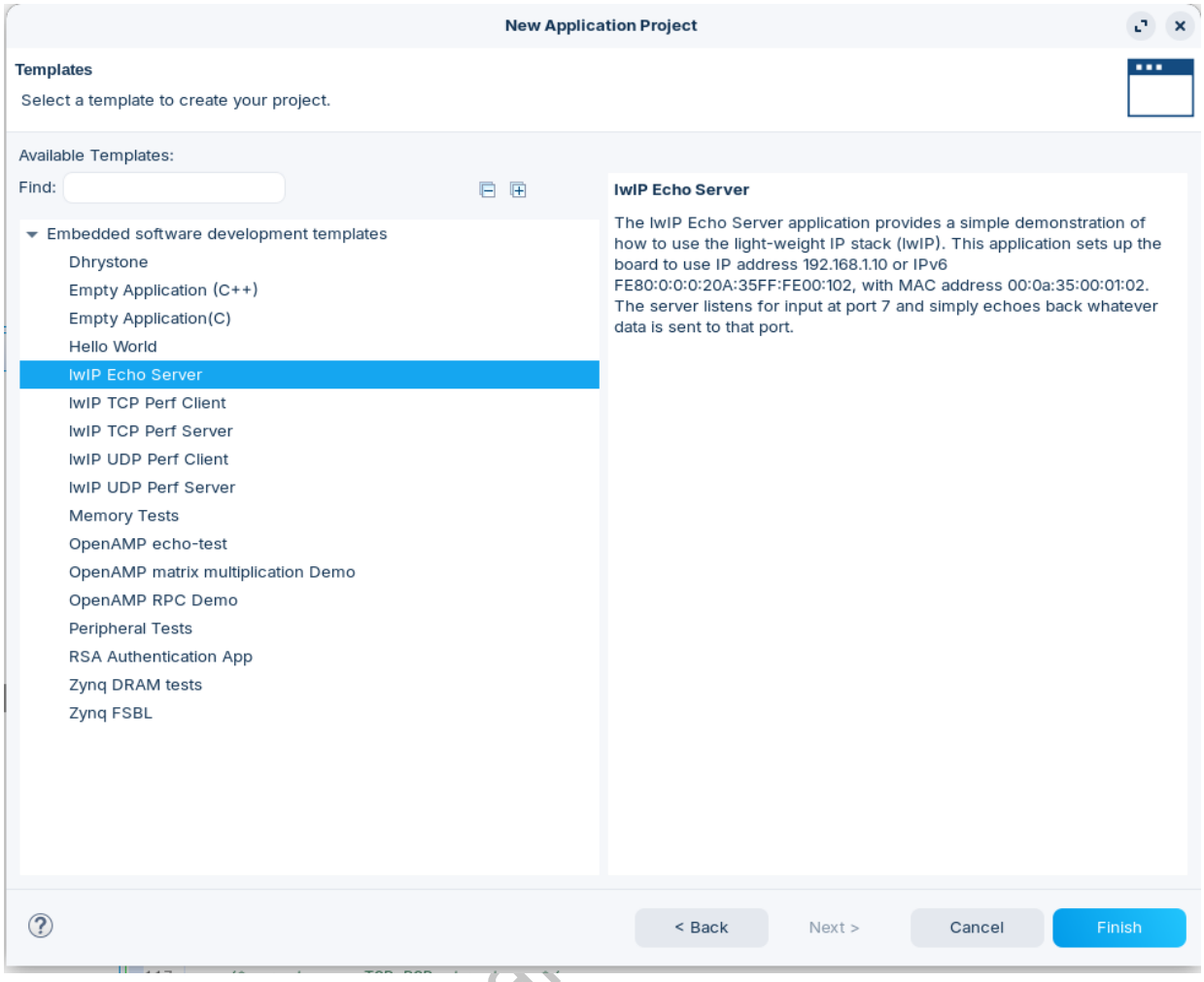
Pasos previos

Lo primero que tienes que tenes el Ethernet configurado en la Zynq, y salvo que estés usando EMIOs para conectar el Ethernet, a éste no se le ven los pines externos que conectar el Ethernet.



Cuando tengas el Bitstream y transfieras el trabajo de Vivado a Vitis creando una plataforma, Vitis te permite partir de un proyecto de ejemplo.

https://soceame.wordpress.com/2024/06/23/como-conectar-una-zynq-por-ethernet/

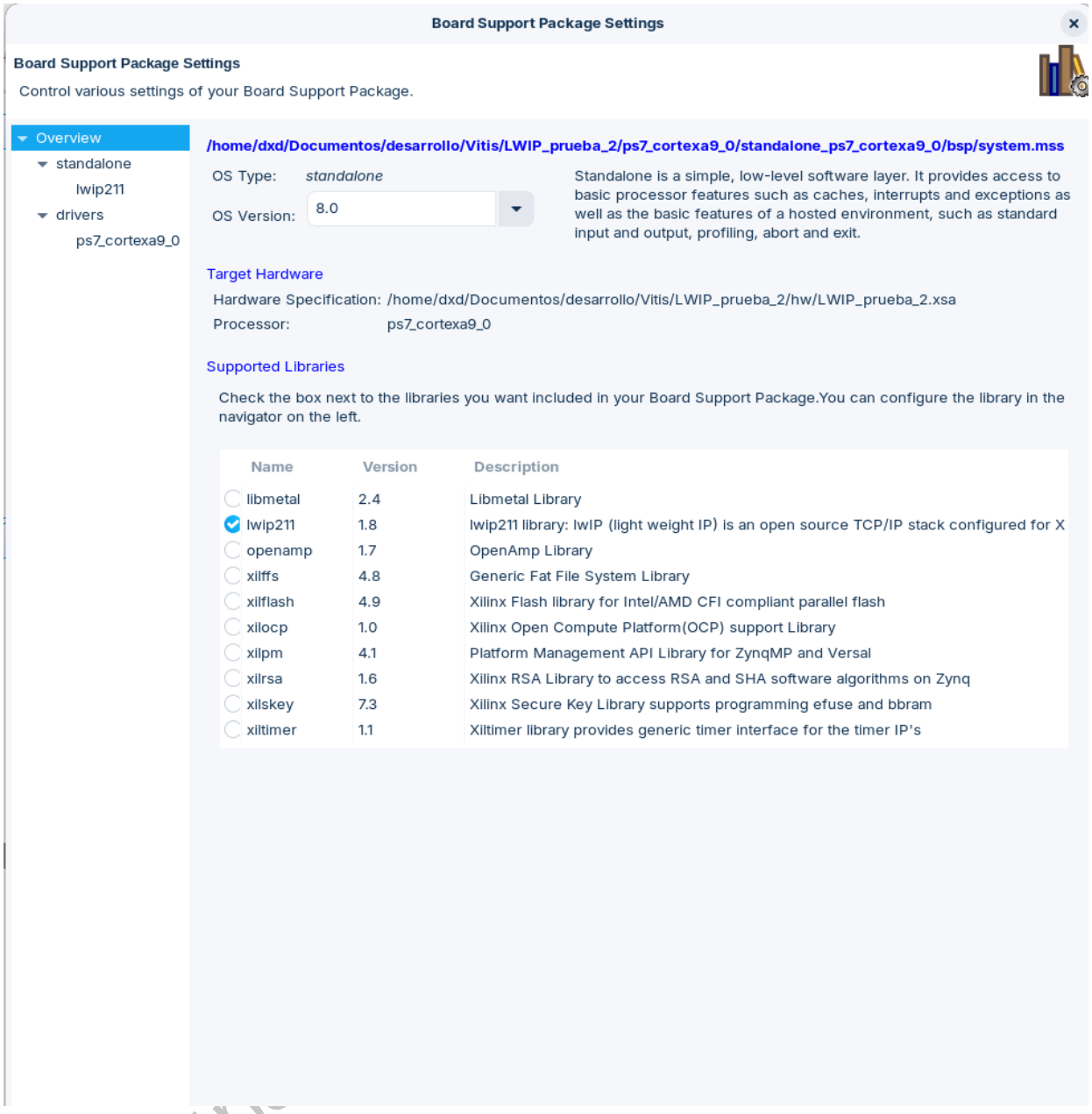


El ejemplo más básico es el «*lwip Echo Server*«, que simplemente es un servidor que recibe un mensaje por Ethernet y lo devuelve.

Desarrollo

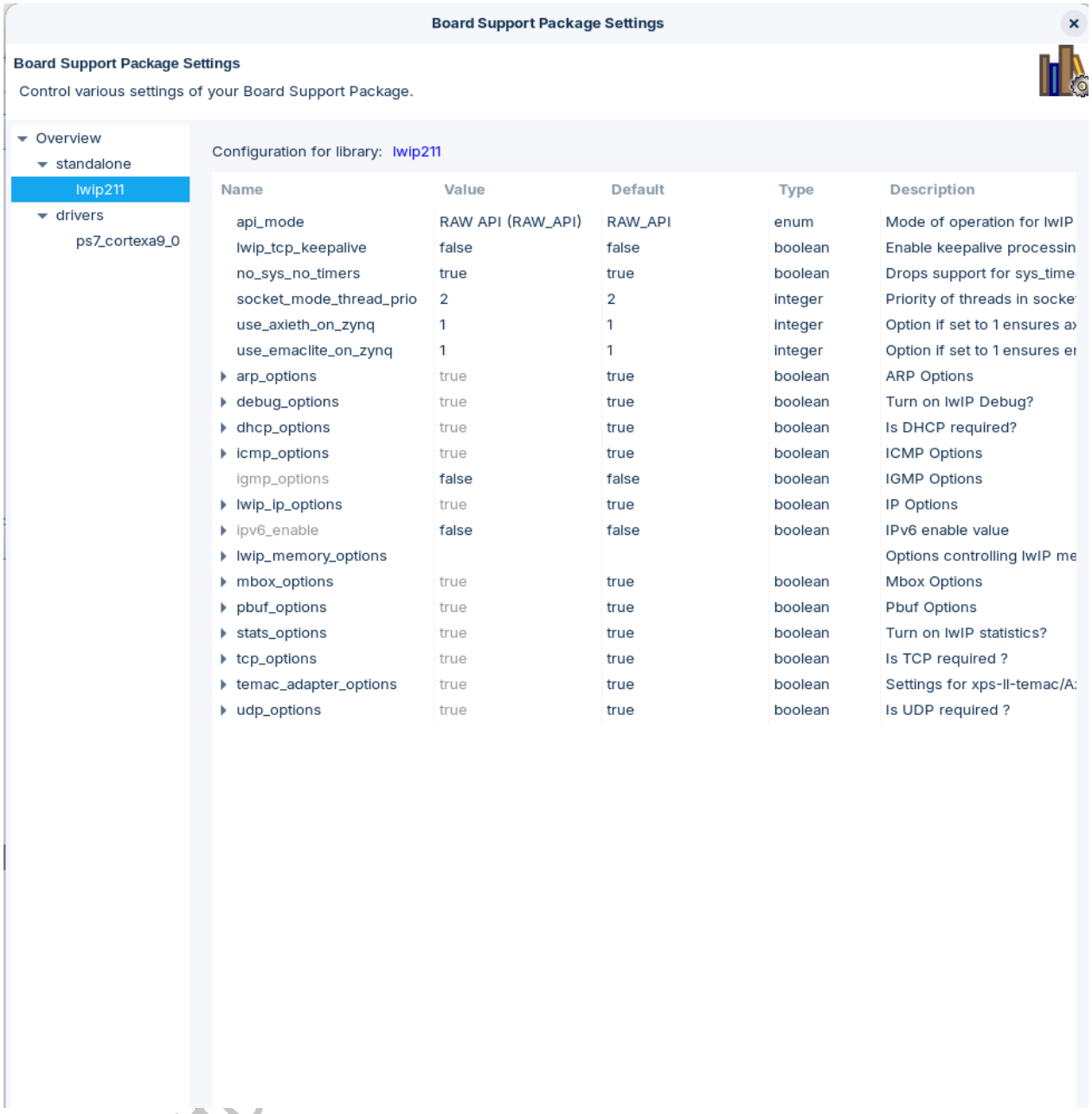
Una vez tengas la librería lwip configurada en el **Board Support Package** (BSP) aparece marcada la casilla de LWIP con la versión de LWIP que utiliza Vitis.

https://soceame.wordpress.com/2024/06/23/como-conectar-una-zynq-por-ethernet/

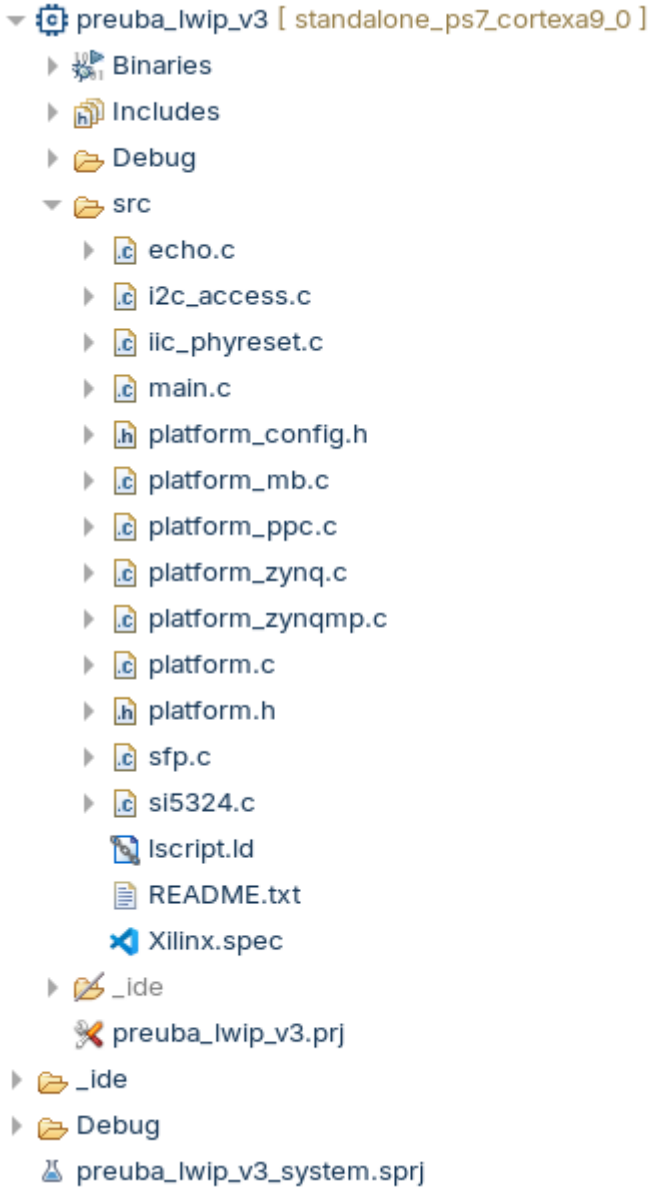


Dentro de esta librería se pueden configurar diferentes parámetros, como por ejemplo que el IP venga por un servidor DHCP.

https://soceame.wordpress.com/2024/06/23/como-conectar-una-zynq-por-ethernet/

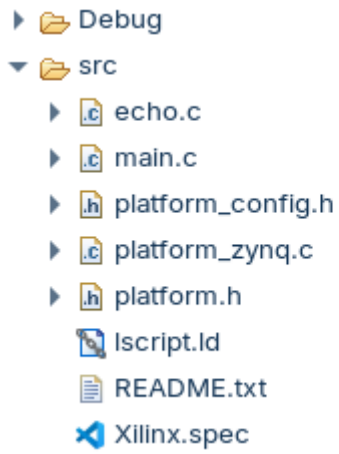


Si no se requiere de tocar nada del ejemplo base, en el proyecto se puede ver la estructura de los ficheros.



En este estructura hay ficheros de más, los cuales se pueden eliminar sin problema alguno.

Por ejemplo, para una Zynq-7000 la estructura se puede quedar de la siguiente forma (también puede ser necesario comentar alguna librería en la que aparezca alguna librería borrada)



Una vez ellegida la estructura, el fichero que contiene toda la información de desarrollo es el **echo.c**.

La función principal y de arranque en este proyecto es la *start_application*, que es llamada desde el **main.c**.

```
int start_application()
{
    struct tcp_pcb *pcb;
    err_t err;
    unsigned port = 7;
    int Status;

    /* create new TCP PCB structure */
    pcb = tcp_new_ip_type(IPADDR_TYPE_ANY);
    if (!pcb) {
        xil_printf("Error creating PCB. Out of Memory\n\r");
        return -1;
    }

    /* bind to specified @port */
    err = tcp_bind(pcb, IP_ANY_TYPE, port);
    if (err != ERR_OK) {
        xil_printf("Unable to bind to port %d: err = %d\n\r", port, err);
        return -2;
    }

    /* we do not need any arguments to callback functions */
    tcp_arg(pcb, NULL);

    /* listen for connections */
    pcb = tcp_listen(pcb);
    if (!pcb) {
        xil_printf("Out of memory while tcp_listen\n\r");
        return -3;
    }

    /* specify callback to use for incoming connections */
    tcp_accept(pcb, accept_callback);

    xil_printf("TCP echo server started @ port %d\n\r", port);

    return 0;
}
```

Esta función une el IP con el puerto (en el ejemplo el 7). Pero también, define cuál es la función de interrupción para cuando llegue un paquete por Ethernet, en el ejemplo se define la función *accept_callback* como la función de interrupción.

Esta función llama a otra función que es la que contiene la implementación de la devolución del dato, que es *recv_callback*.

```
err_t accept_callback(void *arg, struct tcp_pcb *newpcb, err_t err)
{
    static int connection = 1;

    /* set the receive callback for this connection */
    tcp_recv(newpcb, recv_callback);

    /* just use an integer number indicating the connection id as the
       callback argument */
    tcp_arg(newpcb, (void*)(UINTPTR)connection);

    /* increment for subsequent accepted connections */
    connection++;

    return ERR_OK;
}
```

Esta función se ejecuta cuando llega un mensaje al IP y puerto del SoC.

```
err_t recv_callback(void *arg, struct tcp_pcb *tpcb,
                            struct pbuf *p, err_t err)
{

    /* do not read the packet if we are not in ESTABLISHED state */
    if (!p) {
        tcp_close(tpcb);
        tcp_recv(tpcb, NULL);
        return ERR_OK;
    }

    /* indicate that the packet has been received */
    tcp_recved(tpcb, p->len);

    /* echo back the payload */
    /* in this case, we assume that the payload is < TCP_SND_BUF */
    if (tcp_sndbuf(tpcb) > p->len) {
        err = tcp_write(tpcb, p->payload, p->len, 1);

    } else
        xil_printf("no space in tcp_sndbuf\n\r");

    /* free the received pbuf */
    pbuf_free(p);

    return ERR_OK;
}
```

Aquí aparecen dos variables, **tcp**, que es la que contiene la estructura del controlador de Ethernet (definida en la función *start_application*, y **p**, que es la variable que contiene la información sobre el dato que ha entrado.

Tiene dos campos útiles:

- **p->payload:** este campo contiene la dirección de memoria donde está el dato que se he recibido por Ethernet
- **p->len:** este campo contiene el número de bytes que se ha recibido por Ethernet

Bien, pues con esos dos campos se puede ejecutar la aplicación que se desee.

Y por último, aparece la función *tcp_write(…)* que es la función encargada de mandar el dato de vuelta. Para ello necesita 3 campos: *la estructura* en la que se quiere mandar el dato (no tiene porque ser la misma en la que se ha recibido, por ejemplo si se quiere mandar un dato por otro IP). *La dirección de memoria* donde se guarda el dato que se quiere mandar. Y *la longitud* del dato que se quiere mandar.

Con esto puede hacer el juego con la librería LWIP.

Otros parámetros configurables

El *IP* se puede configurar en esta estructura del **main.c**.

```
#if (LWIP_IPV6 == 0)
#if (LWIP_DHCP==1)
    /* Create a new DHCP client for this interface.
     * Note: you must call dhcp_fine_tmr() and dhcp_coarse_tmr() at
     * the predefined regular intervals after starting the client.
     */
    dhcp_start(echo_netif);
    dhcp_timoutcntr = 24;

    while(((echo_netif->ip_addr.addr) == 0) && (dhcp_timoutcntr > 0))
        xemacif_input(echo_netif);

    if (dhcp_timoutcntr <= 0) {
        if ((echo_netif->ip_addr.addr) == 0) {
            xil_printf("DHCP Timeout\r\n");
            xil_printf("Configuring default IP of 192.168.1.10\r\n");
            IP4_ADDR(&(echo_netif->ip_addr),  192, 168,   1, 10);
            IP4_ADDR(&(echo_netif->netmask), 255, 255, 255,  0);
            IP4_ADDR(&(echo_netif->gw),      192, 168,   1,  1);
        }
    }

    ipaddr.addr = echo_netif->ip_addr.addr;
    gw.addr = echo_netif->gw.addr;
    netmask.addr = echo_netif->netmask.addr;
#endif
```

Y la *MAC* del dispositivo en esta parte.

```
/* the mac address of the board. this should be unique per board */
unsigned char mac_ethernet_address[] =
{ 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 };

echo_netif = &server_netif;
```