**Newewww**

#include <stdio.h>﻿#include "xparameters.h"﻿#include "netif/xadapter.h"﻿#include "platform.h"﻿#include "platform\_config.h"﻿#include "xil\_printf.h"﻿#include "xil\_cache.h"﻿// lwIP Includes﻿#include "lwip/init.h"﻿#include "lwip/netif.h"﻿#include "lwip/inet.h"﻿#if LWIP\_DHCP﻿#include "lwip/dhcp.h"﻿extern volatile int dhcp\_timoutcntr;﻿#endif﻿// PTP Includes﻿#include "ptp/ptpd.h"﻿// Hardware Driver Includes﻿#include "xintc.h"﻿#include "xtmrctr.h"﻿// --- Constant Definitions ---﻿#define DEFAULT\_IP\_ADDRESS  "192.168.1.10"﻿#define DEFAULT\_IP\_MASK     "255.255.255.0"﻿#define DEFAULT\_GW\_ADDRESS  "192.168.1.1"﻿#define INTC\_DEVICE\_ID      XPAR\_INTC\_0\_DEVICE\_ID﻿#define TMRCTR\_DEVICE\_ID    XPAR\_TMRCTR\_0\_DEVICE\_ID﻿#define TIMER\_IRPT\_INTR     XPAR\_INTC\_0\_TMRCTR\_0\_VEC\_ID﻿#define PTP\_TICK\_RATE\_HZ    10﻿#define TIMER\_RESET\_VALUE   (XPAR\_AXI\_TIMER\_0\_CLOCK\_FREQ\_HZ / PTP\_TICK\_RATE\_HZ)﻿// --- Global Variables ---﻿struct netif server\_netif;﻿static XIntc interrupt\_controller;﻿static XTmrCtr timer\_controller;﻿volatile int ptp\_timer\_flag = 0;﻿ptp\_clock\_t ptp\_clock;﻿ptpd\_opts ptp\_opts;﻿// --- Function Prototypes ---﻿static int setup\_interrupt\_system();﻿static void Timer\_ISR\_Handler(void \*CallBackRef, u8 TmrCtrNumber);﻿static void ptpd\_opts\_init();﻿static void assign\_default\_ip(ip\_addr\_t \*ip, ip\_addr\_t \*mask, ip\_addr\_t \*gw);﻿int main()﻿{﻿    struct netif \*netif = &server\_netif;﻿    unsigned char mac\_ethernet\_address[] = { 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 };﻿    init\_platform();﻿    xil\_printf("Platform Initialized.\r\n"); // CHECKPOINT 1﻿    lwip\_init();﻿    xil\_printf("lwIP Initialized.\r\n"); // CHECKPOINT 2﻿    if (!xemac\_add(netif, NULL, NULL, NULL, mac\_ethernet\_address, PLATFORM\_EMAC\_BASEADDR)) {﻿        xil\_printf("Error adding network interface\r\n");﻿        return -1;﻿    }﻿    xil\_printf("Network Interface Added.\r\n"); // CHECKPOINT 3﻿    netif\_set\_default(netif);﻿    xil\_printf("Netif is Default.\r\n"); // CHECKPOINT 4﻿    platform\_enable\_interrupts();﻿    xil\_printf("Platform Interrupts Enabled.\r\n"); // CHECKPOINT 5﻿    if (setup\_interrupt\_system() != XST\_SUCCESS) {﻿         xil\_printf("Error setting up interrupts.\r\n");﻿         return -1;﻿    }﻿    xil\_printf("Interrupt System Setup Complete.\r\n"); // CHECKPOINT 6﻿    ptpd\_hw\_timer\_init();﻿    xil\_printf("PTP Hardware Timer Initialized.\r\n"); // CHECKPOINT 7﻿    netif\_set\_up(netif);﻿    xil\_printf("Netif is Up.\r\n"); // CHECKPOINT 8﻿    assign\_default\_ip(&(netif->ip\_addr), &(netif->netmask), &(netif->gw));﻿    xil\_printf("IP Address Assigned: %s\r\n", ip4addr\_ntoa(&(netif->ip\_addr))); // CHECKPOINT 9﻿    ptpd\_opts\_init();﻿    xil\_printf("PTP Options Initialized.\r\n"); // CHECKPOINT 10﻿    ptpd\_net\_init(&ptp\_clock);﻿    xil\_printf("PTP Network Initialized.\r\n"); // CHECKPOINT 11﻿    to\_state(&ptp\_clock, PTP\_INITIALIZING);﻿    xil\_printf("PTP State Machine Started. Entering main loop...\r\n"); // CHECKPOINT 12﻿    while (1) {﻿        xemacif\_input(netif);﻿        if (ptp\_timer\_flag) {﻿            ptp\_timer\_flag = 0;﻿            ptpd\_periodic\_handler();﻿        }﻿    }﻿    cleanup\_platform();﻿    return 0;﻿}﻿// --- All other helper functions remain the same ---﻿static void ptpd\_opts\_init() {﻿    memset(&ptp\_opts, 0, sizeof(ptpd\_opts));﻿    ptp\_opts.slave\_only = TRUE;﻿    ptp\_opts.sync\_interval = 1;﻿    ptp\_opts.announce\_interval = 1;﻿    ptp\_opts.clock\_quality.clock\_class = 255;﻿    ptp\_opts.clock\_quality.clock\_accuracy = 0xFE;﻿    ptp\_opts.clock\_quality.offset\_scaled\_log\_variance = 0xFFFF;﻿    ptp\_opts.priority1 = 255;﻿    ptp\_opts.priority2 = 255;﻿}﻿void Timer\_ISR\_Handler(void \*CallBackRef, u8 TmrCtrNumber) {﻿    ptp\_timer\_flag = 1;﻿}﻿static int setup\_interrupt\_system() {﻿    int status;﻿    status = XIntc\_Initialize(&interrupt\_controller, INTC\_DEVICE\_ID);﻿    if (status != XST\_SUCCESS) return XST\_FAILURE;﻿    status = XTmrCtr\_Initialize(&timer\_controller, TMRCTR\_DEVICE\_ID);﻿    if (status != XST\_SUCCESS) return XST\_FAILURE;﻿    status = XIntc\_Connect(&interrupt\_controller, TIMER\_IRPT\_INTR,﻿                           (XInterruptHandler)XTmrCtr\_InterruptHandler,﻿                           &timer\_controller);﻿    if (status != XST\_SUCCESS) return XST\_FAILURE;﻿    status = XIntc\_Start(&interrupt\_controller, XIN\_REAL\_MODE);﻿    if (status != XST\_SUCCESS) return XST\_FAILURE;﻿    XIntc\_Enable(&interrupt\_controller, TIMER\_IRPT\_INTR);﻿    XTmrCtr\_SetHandler(&timer\_controller, Timer\_ISR\_Handler, NULL);﻿    XTmrCtr\_SetOptions(&timer\_controller, 0, XTC\_INT\_MODE\_OPTION | XTC\_AUTO\_RELOAD\_OPTION);﻿    XTmrCtr\_SetResetValue(&timer\_controller, 0, TIMER\_RESET\_VALUE);﻿    XTmrCtr\_Start(&timer\_controller, 0);﻿    return XST\_SUCCESS;﻿}﻿static void assign\_default\_ip(ip\_addr\_t \*ip, ip\_addr\_t \*mask, ip\_addr\_t \*gw) {﻿    inet\_aton(DEFAULT\_IP\_ADDRESS, ip);﻿    inet\_aton(DEFAULT\_IP\_MASK, mask);﻿    inet\_aton(DEFAULT\_GW\_ADDRESS, gw);﻿}