**This**

The Most Likely Cause: Vivado Block Design﻿The problem is very likely in your hardware design. Please go back to your Vivado Block Design and check this one critical connection:﻿Find your AXI Ethernet Lite IP core.﻿Find its output pin named interrupt.﻿Follow the wire from this pin. It must be connected to your AXI Interrupt Controller (axi\_intc\_0).﻿If you have multiple interrupts, they are usually combined with a Concat IP block first, and then the output of the Concat block goes to the interrupt controller.﻿Here is an example of a correct connection:﻿Please verify that the interrupt from your Ethernet MAC is correctly wired to the interrupt controller in your Vivado design. If it is not, you will need to correct the connection, re-generate the bitstream, and export the hardware to Vitis again. This is the most common cause of the exact problem you are facing.﻿#include "xparameters.h"﻿#include "netif/xadapter.h"﻿#include "platform.h"﻿#include "xil\_printf.h"﻿#include "xil\_cache.h"﻿// lwIP Includes﻿#include "lwip/init.h"﻿#include "lwip/netif.h"﻿#include "lwip/inet.h"﻿// PTP Includes﻿#include "ptp/ptpd.h"﻿// Hardware Driver Includes for Timer and Interrupts﻿#include "xintc.h"﻿#include "xtmrctr.h"﻿// --- Constant Definitions ---﻿#define IP\_ADDRESS          "192.168.0.10"﻿#define NETMASK             "255.255.255.0"﻿#define GATEWAY             "192.168.0.1"﻿#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 Globals (defined here, used by other modules via 'extern')﻿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 print\_ip\_settings(ip\_addr\_t \*ip, ip\_addr\_t \*mask, ip\_addr\_t \*gw);﻿int main()﻿{﻿    struct netif \*netif = &server\_netif;﻿    struct ip\_addr ipaddr, netmask, gw;﻿    /\* The mac address of the board. This should be unique per board \*/﻿    unsigned char mac\_ethernet\_address[] = { 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 };﻿    if (init\_platform() < 0) {﻿        xil\_printf("ERROR initializing platform.\r\n");﻿        return -1;﻿    }﻿    xil\_printf("\r\n--- PTP Bare-Metal Application ---\r\n");﻿    /\* Initialize IP addresses to be used \*/﻿    inet\_aton(IP\_ADDRESS, &ipaddr);﻿    inet\_aton(NETMASK, &netmask);﻿    inet\_aton(GATEWAY, &gw);﻿    print\_ip\_settings(&ipaddr, &netmask, &gw);﻿    lwip\_init();﻿    /\* Add network interface to the netif\_list, and set it as default \*/﻿    if (!xemac\_add(netif, &ipaddr, &netmask, &gw, mac\_ethernet\_address, PLATFORM\_EMAC\_BASEADDR)) {﻿        xil\_printf("Error adding N/W interface\r\n");﻿        return -1;﻿    }﻿    netif\_set\_default(netif);﻿    /\* Specify that the network interface is up \*/﻿    netif\_set\_up(netif);﻿    /\* Now enable interrupts and set up our timers \*/﻿    platform\_enable\_interrupts();﻿    setup\_interrupt\_system(); // For the periodic PTP tick﻿    ptpd\_hw\_timer\_init();     // For the 64-bit high-res timestamp﻿    /\* Initialize the PTPd stack \*/﻿    ptpd\_opts\_init();﻿    ptpd\_net\_init(&ptp\_clock);﻿    to\_state(&ptp\_clock, PTP\_INITIALIZING); // Start the PTP state machine﻿    xil\_printf("PTP initialized. Entering main loop...\r\n");﻿    /\* Main application loop \*/﻿    while (1) {﻿        // Poll the network interface for incoming packets﻿        xemacif\_input(netif);﻿        // Check if the periodic timer has fired to run the PTP handler﻿        if (ptp\_timer\_flag) {﻿            ptp\_timer\_flag = 0; // Reset the flag﻿            ptpd\_periodic\_handler(); // This calls protocol\_tick()﻿        }﻿    }﻿    /\* Never reached \*/﻿    cleanup\_platform();﻿    return 0;﻿}﻿// --- Initialization and Helper Functions ---﻿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 print\_ip(char \*msg, ip\_addr\_t \*ip)﻿{﻿    xil\_printf("%s: %d.%d.%d.%d\r\n", msg,﻿               ip4\_addr1(ip), ip4\_addr2(ip), ip4\_addr3(ip), ip4\_addr4(ip));﻿}﻿static void print\_ip\_settings(ip\_addr\_t \*ip, ip\_addr\_t \*mask, ip\_addr\_t \*gw)﻿{﻿    print\_ip("Board IP", ip);﻿    print\_ip("Netmask", mask);﻿    print\_ip("Gateway", gw);﻿}