```
/////////////////////////////////////////////////////////////
// File - fpga_diag.c
//
// This is a diagnostics application for accessing the FPGA card.
// Code was generated by DriverWizard v6.01 - http://www.jungo.com.
// The library accesses the hardware via WinDriver functions.
// It accesses the hardware via WinDriver functions.
//
// Copyright (c) 2003 Jungo Ltd.  http://www.jungo.com
//
/////////////////////////////////////////////////////////////

#include "fpga_lib.h"
#include "c:/windriver/samples/shared/pci_diag_lib.h"
#include <stdio.h>
#include "c:/windriver/include/status_strings.h"

// input command from user
static char line[256];

char *FPGA_GetAddrSpaceName(FPGA_ADDR addrSpace)
{
    return
        addrSpace==FPGA_AD_BAR0 ? "Addr Space BAR0" :
        addrSpace==FPGA_AD_BAR1 ? "Addr Space BAR1" :
        addrSpace==FPGA_AD_BAR2 ? "Addr Space BAR2" :
        addrSpace==FPGA_AD_BAR3 ? "Addr Space BAR3" :
        addrSpace==FPGA_AD_BAR4 ? "Addr Space BAR4" :
        addrSpace==FPGA_AD_BAR5 ? "Addr Space BAR5" :
        addrSpace==FPGA_AD_EPROM ? "EEPROM Addr Space" :
        "Invalid";
}

void FPGA_AccessRanges(FPGA_HANDLE hFPGA)
{
    int cmd, cmd2, i;
    UINT addr, data = 0;
    FPGA_ADDR ad_sp = 0;
    FPGA_MODE ad_mode = FPGA_MODE_DWORD;

    for (; ad_sp<FPGA_ITEMS && !FPGA_IsAddrSpaceActive(hFPGA, ad_sp); ad_sp++)
    if (ad_sp==FPGA_ITEMS)
    {
        printf ("No active memory or IO ranges on board!¥n");
        return;
    }

    do
    {
        printf ("Access the board's memory and IO ranges¥n");
        printf ("-----------------------------------¥n");
        printf ("1. Change active memory space: %s¥n",FPGA_GetAddrSpaceName
(ad_sp));
```

```
printf ("2. Toggle active mode: %s¥n",
    ad_mode==FPGA_MODE_BYTE ? "BYTE (8 bit)" :
    ad_mode==FPGA_MODE_WORD ? "WORD (16 bit)" : "DWORD (32 bit)");
printf ("3. Read from board¥n");
printf ("4. Write to board¥n");
printf ("99. Back to main menu¥n");
printf ("¥n");
printf ("Enter option: ");
cmd = 0;
fgets(line, sizeof(line), stdin);
sscanf (line, "%d",&cmd);
switch (cmd)
{
case 1:
    printf ("Choose memory or IO space:¥n");
    printf ("--------------------------¥n");
    for (i=0; i<FPGA_ITEMS; i++)
    {
        printf ("%d. %s", i+1, FPGA_GetAddrSpaceName(i));
        if (!FPGA_IsAddrSpaceActive(hFPGA, (FPGA_ADDR) i))
            printf (" - space not active");
        printf ("¥n");
    }
    printf ("Enter option: ");
    cmd2 = 99;
    fgets(line, sizeof(line), stdin);
    sscanf (line, "%d",&cmd2);
    if (cmd2>=1 && cmd2<FPGA_ITEMS+1)
    {
        ad_sp = cmd2-1;
        if (!FPGA_IsAddrSpaceActive(hFPGA, ad_sp))
            printf ("Chosen space not active!¥n");
    }
    break;
case 2:
    ad_mode = (ad_mode + 1) % 3;
    break;
case 3:
    printf ("Enter offset to read from: ");
    fgets(line, sizeof(line), stdin);
    sscanf (line, "%x", &addr);
    switch (ad_mode)
    {
    case FPGA_MODE_BYTE:
        data = FPGA_ReadByte(hFPGA, ad_sp, addr);
        break;
    case FPGA_MODE_WORD:
        data = FPGA_ReadWord(hFPGA, ad_sp, addr);
        break;
    case FPGA_MODE_DWORD:
        data = FPGA_ReadDword(hFPGA, ad_sp, addr);
        break;
    }
```

```
            printf ("Value read: %x¥n", data);
            break;
        case 4:
            printf ("Enter offset to write to: ");
            fgets(line, sizeof(line), stdin);
            sscanf (line,"%x", &addr);
            printf ("Enter data to write %s: ",
                ad_mode==FPGA_MODE_BYTE ? "BYTE (8 bit)" :
                ad_mode==FPGA_MODE_WORD ? "WORD (16 bit)" : "DWORD (32 bit)");
            fgets(line, sizeof(line), stdin);
            sscanf (line, "%x",&data);
            switch (ad_mode)
            {
            case FPGA_MODE_BYTE:
                FPGA_WriteByte(hFPGA, ad_sp, addr, (BYTE) data);
                break;
            case FPGA_MODE_WORD:
                FPGA_WriteWord(hFPGA, ad_sp, addr, (WORD) data);
                break;
            case FPGA_MODE_DWORD:
                FPGA_WriteDword(hFPGA, ad_sp, addr, data);
                break;
            }
            break;
        }
    } while (cmd!=99);
}

void FPGA_IntHandlerRoutine(FPGA_HANDLE hFPGA, FPGA_INT_RESULT *intResult)
{
    printf ("Got Int number %u¥n", (UINT)intResult->dwCounter);
}

void FPGA_EnableDisableInterrupts(FPGA_HANDLE hFPGA)
{
    int cmd;

    printf ("WARNING!!!¥n");
    printf ("----------¥n");
    printf ("Your hardware has level sensitive interrupts.¥n");
    printf ("You must modify the source code of FPGA_IntEnable(), in the file
fpga_lib.c,¥n");
    printf ("to acknowledge the interrupt before enabling interrupts.¥n");
    printf ("Without this modification, your PC will HANG upon interrupt!¥n");
    printf ("¥n");

    do
    {
        printf ("Enable / Disable interrupts¥n");
        printf ("--------------------------¥n");
        printf ("1. %s Int¥n", FPGA_IntIsEnabled(hFPGA) ? "Disable" : "Enable");
        printf ("99. Back to main menu¥n");
        printf ("¥n");
```

```
        printf ("Enter option: ");
        cmd = 0;
        fgets(line, sizeof(line), stdin);
        sscanf (line, "%d",&cmd);
        switch (cmd)
        {
        case 1:
            if (FPGA_IntIsEnabled(hFPGA))
            {
                printf ("Disabling interrupt Int¥n");
                FPGA_IntDisable(hFPGA);
            }
            else
            {
                printf ("Enabling interrupt Int¥n");
                if (!FPGA_IntEnable(hFPGA, FPGA_IntHandlerRoutine))
                    printf ("%s", FPGA_ErrorString);
            }
            break;
        }
    } while (cmd!=99);
}

void FPGA_HandlerFunc(WD_EVENT *event, void *data)
{
    printf("¥nreceive notification: ");
    switch (event->dwAction)
    {
    case WD_INSERT:
        printf("WD_INSERT¥n");
        break;
    case WD_REMOVE:
        printf("WD_REMOVE¥n");
        break;
    case WD_POWER_CHANGED_D0:
        printf("WD_POWER_CHANGED_D0¥n");
        break;
    case WD_POWER_CHANGED_D1:
        printf("WD_POWER_CHANGED_D1¥n");
        break;
    case WD_POWER_CHANGED_D2:
        printf("WD_POWER_CHANGED_D2¥n");
        break;
    case WD_POWER_CHANGED_D3:
        printf("WD_POWER_CHANGED_D3¥n");
        break;
    case WD_POWER_SYSTEM_WORKING:
        printf("WD_POWER_SYSTEM_WORKING¥n");
        break;
    case WD_POWER_SYSTEM_SLEEPING1:
        printf("WD_POWER_SYSTEM_SLEEPING1¥n");
        break;
    case WD_POWER_SYSTEM_SLEEPING2:
```

```
        printf("WD_POWER_SYSTEM_SLEEPING2¥n");
        break;
    case WD_POWER_SYSTEM_SLEEPING3:
        printf("WD_POWER_SYSTEM_SLEEPING3¥n");
        break;
    case WD_POWER_SYSTEM_HIBERNATE:
        printf("WD_POWER_SYSTEM_HIBERNATE¥n");
        break;
    case WD_POWER_SYSTEM_SHUTDOWN:
        printf("WD_POWER_SYSTEM_SHUTDOWN¥n");
        break;
    }
}

FPGA_HANDLE FPGA_LocateAndOpenBoard(DWORD dwVendorID, DWORD dwDeviceID)
{
    DWORD cards, my_card;
    FPGA_HANDLE hFPGA = NULL;

    if (dwVendorID==0)
    {
        printf ("Enter VendorID: ");
        fgets(line, sizeof(line), stdin);
        sscanf (line, "%x",&dwVendorID);
        if (dwVendorID==0) return NULL;

        printf ("Enter DeviceID: ");
        fgets(line, sizeof(line), stdin);
        sscanf (line, "%x",&dwDeviceID);
    }
    cards = FPGA_CountCards (dwVendorID, dwDeviceID);
    if (cards==0)
    {
        printf ("%s", FPGA_ErrorString);
        return NULL;
    }
    else if (cards==1)
        my_card = 1;
    else
    {
        UINT i;

        printf ("Found %u matching PCI cards¥n", (UINT)cards);
        printf ("Select card (1-%u): ", (UINT)cards);
        i = 0;
        fgets(line, sizeof(line), stdin);
        sscanf (line, "%d",&i);
        if (i>=1 && i <=cards) my_card = i;
        else
        {
            printf ("Choice is out of range¥n");
            return NULL;
        }
```

```
    }
    if (!FPGA_Open (&hFPGA, dwVendorID, dwDeviceID, my_card - 1))
    {
        printf ("%s", FPGA_ErrorString);
        return NULL;
    }
    printf ("FPGA PCI card found!¥n");
    return hFPGA;
}

int main(int argc, char *argv[])
{
    int cmd;
    FPGA_HANDLE hFPGA = NULL;
    HANDLE hWD;
    WD_PCI_SLOT pciSlot;
    DWORD dwAction = 0;
    BOOL fRegisteredEvent = FALSE;

    printf ("FPGA diagnostic utility.¥n");
    printf ("Application accesses hardware using WinDriver.¥n");

    // Make sure WinDriver is loaded
    if (!PCI_Get_WD_handle(&hWD))
        return 0;
    WD_Close (hWD);

    hFPGA = FPGA_LocateAndOpenBoard(FPGA_DEFAULT_VENDOR_ID,
FPGA_DEFAULT_DEVICE_ID);

    do
    {
        printf ("¥n");
        printf ("FPGA main menu¥n");
        printf ("--------------------¥n");
        printf ("1. Scan PCI bus¥n");
        printf ("2. Locate/Choose FPGA board¥n");
        if (hFPGA)
        {
            printf ("3. PCI configuration registers¥n");
            printf ("5. Access FPGA memory and IO ranges¥n");
            printf ("6. Enable / Disable interrupts¥n");
            printf ("7. %s events¥n", fRegisteredEvent ? "Unregister" :
"Register");
        }
        printf ("99. Exit¥n");
        printf ("Enter option: ");
        cmd = 0;
        fgets(line, sizeof(line), stdin);
        sscanf (line, "%d",&cmd);
        switch (cmd)
        {
        case 1: // Scan PCI bus
```

```
            PCI_Print_all_cards_info();
            break;
        case 2: // Locate FPGA board
            if (hFPGA) FPGA_Close(hFPGA);
            hFPGA = FPGA_LocateAndOpenBoard(0, 0);
            if (!hFPGA)
                printf("FPGA card open failed!¥n");
            break;
        case 3: // PCI configuration registers
            if (hFPGA)
            {
                WD_PCI_SLOT pciSlot;
                FPGA_GetPciSlot(hFPGA, &pciSlot);
                PCI_EditConfigReg(pciSlot);
                break;
            }
        case 5: // Access FPGA memory and IO ranges
            if (hFPGA) FPGA_AccessRanges(hFPGA);
            break;
        case 6: // Enable / Disable interrupts
            if (hFPGA)
                FPGA_EnableDisableInterrupts(hFPGA);
            break;
        case 7: // Register / Unregister event
            if (hFPGA && !fRegisteredEvent)
            {
                dwAction |= WD_INSERT | WD_REMOVE;
                dwAction |= WD_ACTIONS_POWER;

                FPGA_GetPciSlot(hFPGA, &pciSlot);

                if (!FPGA_RegisterEvent(hFPGA, dwAction, FPGA_DEFAULT_VENDOR_ID,
                    FPGA_DEFAULT_DEVICE_ID, pciSlot, FPGA_HandlerFunc))
                    printf ("%s", FPGA_ErrorString);
                else
                    fRegisteredEvent = TRUE;
                break;
            }
            if (hFPGA && fRegisteredEvent)
            {
                fRegisteredEvent = FALSE;
                FPGA_UnregisterEvent(hFPGA);
            }
            break;
        }
    } while (cmd!=99);

    if (hFPGA)
        FPGA_Close(hFPGA);

    return 0;
}
```