PCI-EK01 Driver Level Programming Guide

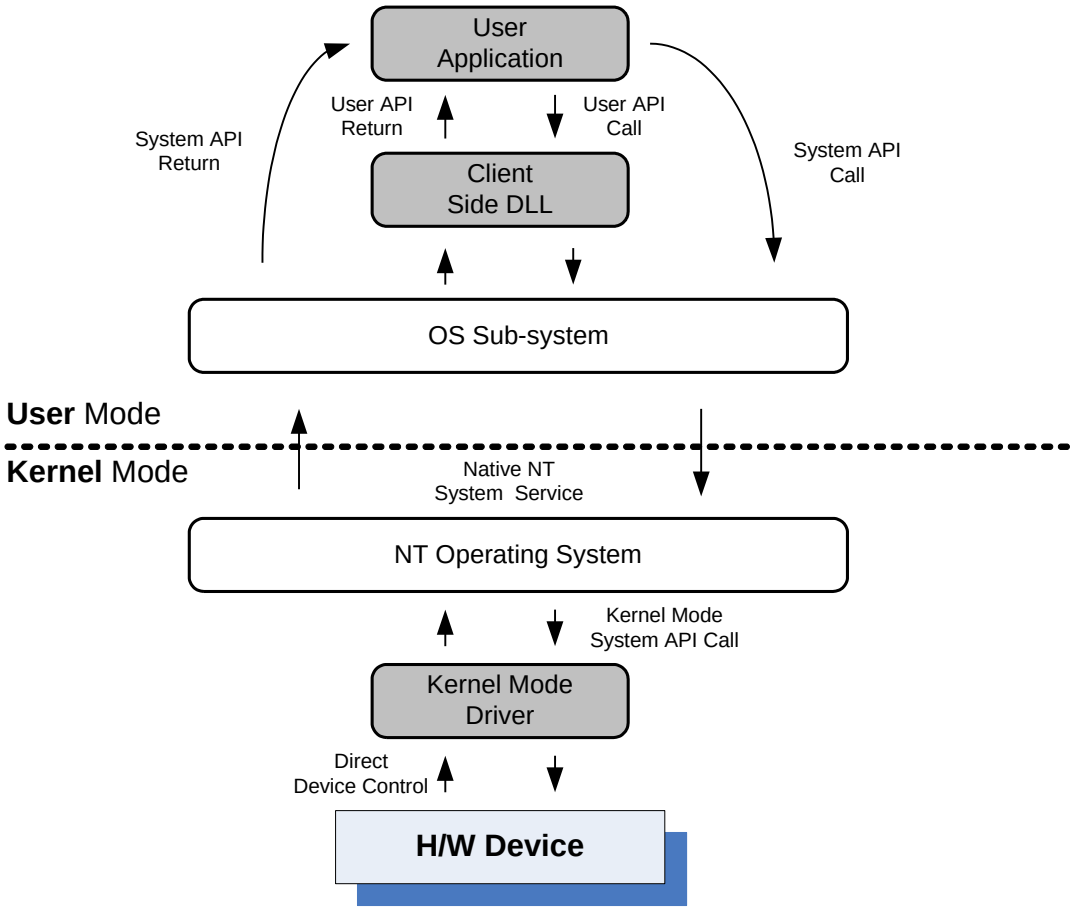
-- Contents --

1. Device Driver

Most PC is currently prohibited to control Hardware by user application. The main reason was running only one program on your PC in the past, but today many programs are performed during the same period of time, it will be needed protecting the system and more stable operation in a multitasking environment.

The old DOS and Windows95, 98, me, the support for multi-tasking, but were also able to access the hardware directly. However, when mounting a new hardware device (PC) on the most operating systems (Unix and Windows NT family-NT4.0 Windows2000, Windows XP), "Kernel Mode Device Driver" must be provided with the Hardware, so you can control the Hardware in your application. For a description of the future is based on the Windows NT operating system.

In the below pictures, an application shows how to access the hardware device and level of hierarchy of any NT OS Kernel.



<Figure 1.1 Windows NT Hardware Device Control>

As shown in Figure 1.1, the case of controlling the hardware devices in the application, always pass through the kernel-mode driver. If you want to use a standard PC devices (mouse, keyboard, display) in basic mode, you do not need another kernel mode driver. However, if no special operation corresponding to the hardware as standard equipment, it is necessary to provide another driver.

DAQ system Co., Ltd provides a kernel-mode device driver that is suitable for the application in the board.

In figure 1.1, there are two methods to send/receive data between Kernel Mode Driver and an application program. The first method is that use the “Client Side Dll”, the second method is that communicates the driver through the system API call.

The "Client Side DLL" is provided by the company that provides the device driver to allow the user to use a hardware device easily and safely. If you use the first method, the speed of application development time can be decreased and a more stable interface can be used. But, you may not control the user needs in a special case, if you have problems, debug the program itself is not easy.

On the other hand, the second way can be controlled more precise the board because user can control register level control. However, in order to use, user need to more understand the manual.

This article describes how the user to communicate with the kernel-mode device driver directly without using the "Client Side Dll".

2. Win32 API(System call)s

Main system API that is used to communicate with hardware devices Win32 is as follows. (Refer to the MSDN Win32 for detailed function description.)

(1) CreateFile

```
HANDLE CreateFile(
  LPCTSTR lpFileName,                    // pointer to name of the file
  DWORD dwDesiredAccess,                 // access (read-write) mode
  DWORD dwShareMode,                     // share mode
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                                         // pointer to security attributes
  DWORD dwCreationDisposition,           // how to create
  DWORD dwFlagsAndAttributes,            // file attributes
  HANDLE hTemplateFile                   // handle to file with attributes to copy
);
```

(2) ReadFile

```
BOOL ReadFile(
  HANDLE hFile,                          // handle of file to read
  LPVOID lpBuffer,                       // pointer to buffer that receives data
  DWORD nNumberOfBytesToRead,            // number of bytes to read
  LPDWORD lpNumberOfBytesRead,           // pointer to number of bytes read
  LPOVERLAPPED lpOverlapped              // pointer to structure for data
);
```

(3) WriteFile

```
BOOL WriteFile(
  HANDLE hFile,                          // handle to file to write to
  LPCVOID lpBuffer,                      // pointer to data to write to file
  DWORD nNumberOfBytesToWrite,           // number of bytes to write
  LPDWORD lpNumberOfBytesWritten,        // pointer to number of bytes written
  LPOVERLAPPED lpOverlapped              // pointer to structure for overlapped I/O
);
```

(4) DeviceIoControl

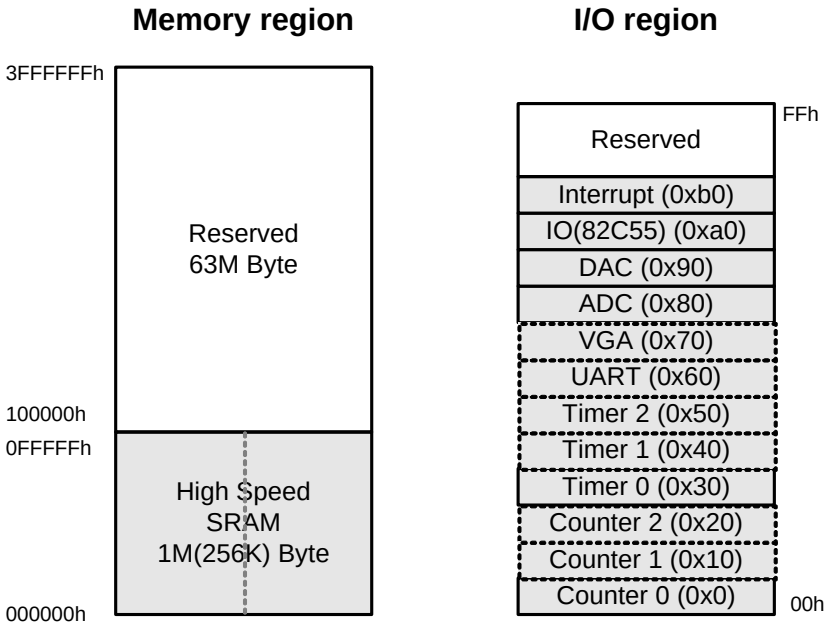
```
BOOL DeviceIoControl(
  HANDLE hDevice,                 // handle to device of interest
  DWORD dwIoControlCode,          // control code of operation to perform
  LPVOID lpInBuffer,              // pointer to buffer to supply input data
  DWORD nInBufferSize,            // size, in bytes, of input buffer
  LPVOID lpOutBuffer,             // pointer to buffer to receive output data
  DWORD nOutBufferSize,           // size, in bytes, of output buffer
  LPDWORD lpBytesReturned,        // pointer to variable to receive byte count
  LPOVERLAPPED lpOverlapped       // pointer to structure for asynchronous operation
);
```

Four functions is for making a device to control Hardware device(Software Perspective), and reading/writing/controlling a data. ReadFile and WriteFile API are not using at PCI-EK01(A/B) board. But, all control and data read/write is implemented only using Device Control.

3. Hardware Resource

A device functions control of PCI=EK01 are located in the memory and I / O area on PCI-EK01. For example, using the I/O area address 0 of the control register is for controlling of the counter 0.

The total allocation received from the system in case of Memory 64Mbyte and I/O 256byte at PCI-EK01. As shown in the figure, does not use any area assigned, reserved area is for future enhancements.



.

(Notice) 1. Refer to “AN203(PCI-EK01 Register Level Application Guide)” for I / O devices, and a detailed description of memory.

(Notice) 2. The dotted line of the figure shows that PCI-EK01(A) are not supported. VGA and UART function plans to add at a later.

4. CreateFile

To control hardware devices in the application, it must get a handle and create a device by using the first CreateFile function. The usage of the function is as follows.

The return value of the function gets the driver handle. The value of the handle later ReadFile, WriteFile, DeviceIoControl function is used as the argument value.

```
char *sLinkName = "₩₩₩₩.₩₩Pci_ek01_0";      // Hardware device name

                                             // Create a handle to the driver
m_hDriver = CreateFile( sLinkName,
            GENERIC_READ | GENERIC_WRITE,
            FILE_SHARE_READ ,
            NULL,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL| FILE_FLAG_OVERLAPPED,
            NULL);

if(m_hDriver == INVALID_HANDLE_VALUE)
{
    MessageBox("error to open driver");
}
```

The argument value ("₩₩₩.₩₩ Pci_ek01_0") as using the name of a symbolic link is a PCI-EK01 device name that registered in the system.

If the system(PC) is equipped with multiple boards, it is registered by increasing in the number of the end part as a Pci_ek01_1.

4. DeviceIoControl

When the Device handle value is gotten by CreateFile function, this handle value is used as an argument of the DiviceIoControl function to control the device. The usage of the function is shown below, detailed function description, see the MSDN.

```
bResult =   DeviceIoControl(    m_hDriver,
                                PCI_EK01_IOCTL_MEM_WRITE,   // Control Code
                                bufInput,
                                sizeof(DWORD) * NumberOfWrite,
                                buf,
                                NumberOfRead,
                                &NumberOfRead,
                                NULL);
```

Used as the second argument of the function value "Control Code" value, this value can be different behavior depending on the use, the main code is as follows.

(1) PCI_EK01_IOCTL_MEM_READ
(2) PCI_EK01_IOCTL_MEM_WRITE
(3) PCI_EK01_IOCTL_IO_READ
(4) PCI_EK01_IOCTL_IO_WRITE

When calling DeviceIoControl by using each control code, the bufInput must have an appropriate value, and the buf (Output Buffer) to store the return value should be enough space. The buffer size is always specified in bytes.

If one Read operation, the first argument is the address Offset, the number of bytes read by the second argument is stored as a 32-bit DWORD value, the output buf is needed to have enough space to store the read data.

If one Write operation, the first argument is the address Offset, the number of bytes write by the second argument is stored as a 32-bit DWORD value, the following is the value of the data to be written.

References

1. MSDN (MicroSoft Developer Network)

-- Microsoft

2. AN203 - PCI-EK01 Register Level Application Guide

-- DAQ system

3. PCI-EK01(A/B) User's Manual

-- DAQ system