PCI-EK01 Driver Level Programming Guide

-- 목차 --

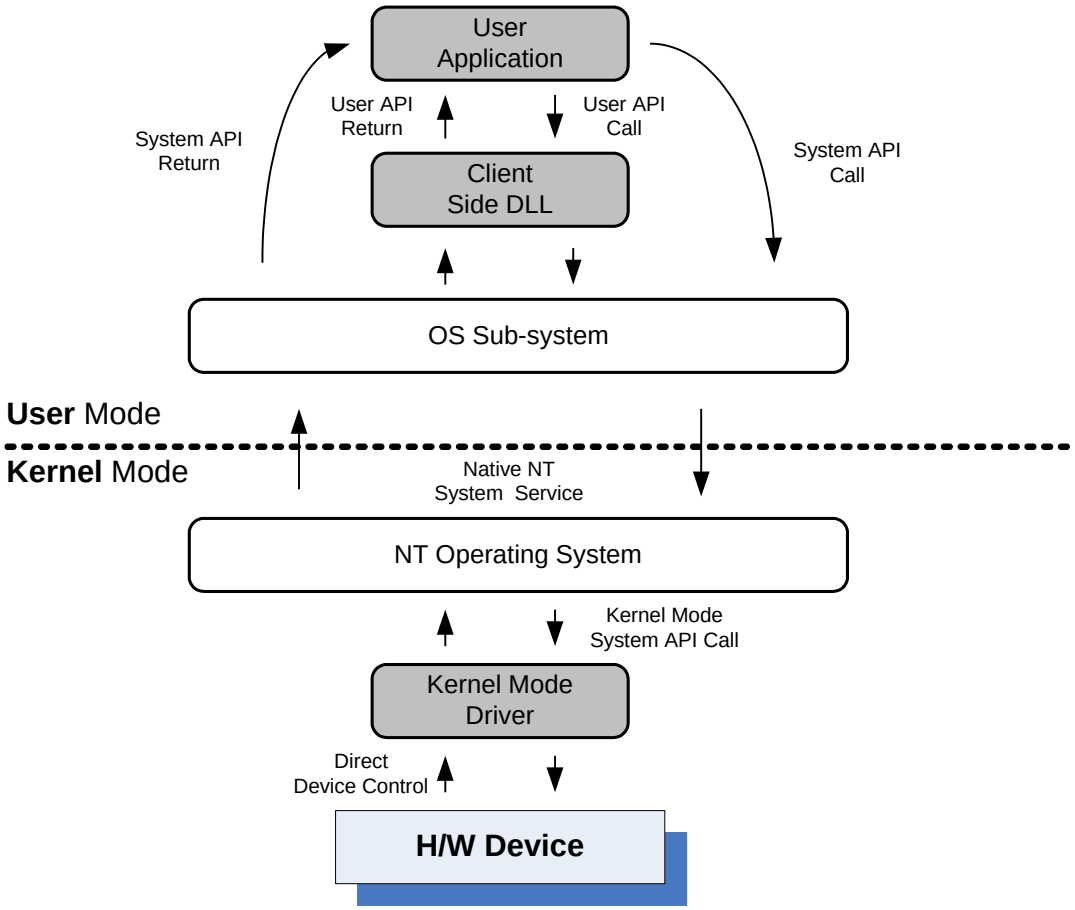
1. Device Driver

현재 대부분의 PC에서 사용하는 운용체제(Operating System)은 사용자 응용 프로그램이 직접 하드웨어를 제어하는 것을 금지하고 있다.

주된 이유는 PC에서 하나의 프로그램만을 실행시키던 예전과는 달리 여러 프로그램을 동시에 실행해야 하는 멀티태스킹 환경에서 시스템을 보호하여 보다 안정된 동작을 하기 위해서이다.

예전 DOS 및 Windows95, 98, me의 경우 멀티태스킹을 지원하였지만, 또한 직접 하드웨어 접근이 가능하였다. 그러나, 대부분의 운용체제(Unix 및 Windows NT계열-NT4.0 Windows2000, Windows XP)에서는 새로운 하드웨어 장치를 시스템(PC)에 장착을 할 경우 반드시 "Kernel Mode Device Driver"를 함께 제공해 줌으로서 하드웨어를 사용자 응용 프로그램에서 제어 할 수 있도록 한다. 앞으로의 설명은 Windows NT 운영체제를 기준으로 설명을 한다.

아래 그림을 보면 응용 프로그램이 NT OS Kernel에서 어떠한 계층 및 단계로 하드웨어 장치에 접근하는 가를 보여주고 있다.



<그림 1.1 Windows NT Hardware Device Control>

그림 1.1에서 보듯이 응용 프로그램에서 하드웨어 장치를 제어할 경우에는 커널 모드 드라이버를 반드시 거치게 되어있다. 표준 PC 장치(마우스, 키보드, 디스플레이)를 기본 모드에서 사용할 경우에는 별도의 커널 모드 드라이버가 필요가 없다. 하지만 표준 장치라고 하여도 하드웨어에 따른 특별한 동작의 경우 별도의 드라이버를 제공할 필요가 있다.

DAQ system에서 제공하는 대부분의 Data Acquisition board는 응용 프로그램에서 보드를 적절히 제어하여 원하는 기능을 얻기 위하여 보드에 맞는 커널 모드 디바이스 드라이버를 제공한다.

그림 1.1에서 커널 모드 드라이버와 응용 프로그램간에 제어 및 데이터를 주고 받기 위해 두 가지 방식을 이용할 수 있다. 첫 번째는 “Client Side Dll”를 이용하는 것과, 두 번째는 직접 system API call을 통해 드라이버와 통신을 하는 것이다.

“Client Side DLL”은 디바이스 드라이버를 제공하는 회사에서 사용자가 쉽고 안전하게 하드웨어 장치를 사용할 수 있도록 제공하므로 첫 번째 방법을 이용할 경우 응용 프로그램의 개발 기간의 단축 및 보다 안정된 인터페이스를 이용할 수 있다. 하지만, 어떠한 특별한 경우 사용자가 요구하는 제어가 가능하지 않을 수 있고 문제가 발생할 경우 프로그램 디버깅 자체가 쉽지 않다.
반면에 두 번째 방법은 사용자가 보드의 레지스터 레벨 제어를 할 수 있으므로 보다 정밀하게 보드를 제어 할 수 있지만 사용을 하기 위하여는 자세한 보드 매뉴얼을 숙지하는 것이 필수라 하겠다.

본 문서는 “Client Side Dll”을 이용하지 않고 사용자가 직접 커널 모드 디바이스 드라이버와 통신하며 제어를 하는 방법을 설명할 것이다.

2. Win32 API(System call)s

Win32에서 하드웨어 장치와의 통신을 위해 사용하는 주요 시스템 API는 다음과 같다. (자세한 함수의 설명은 MSDN의 Win32 부분을 참조하기 바람.)

(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

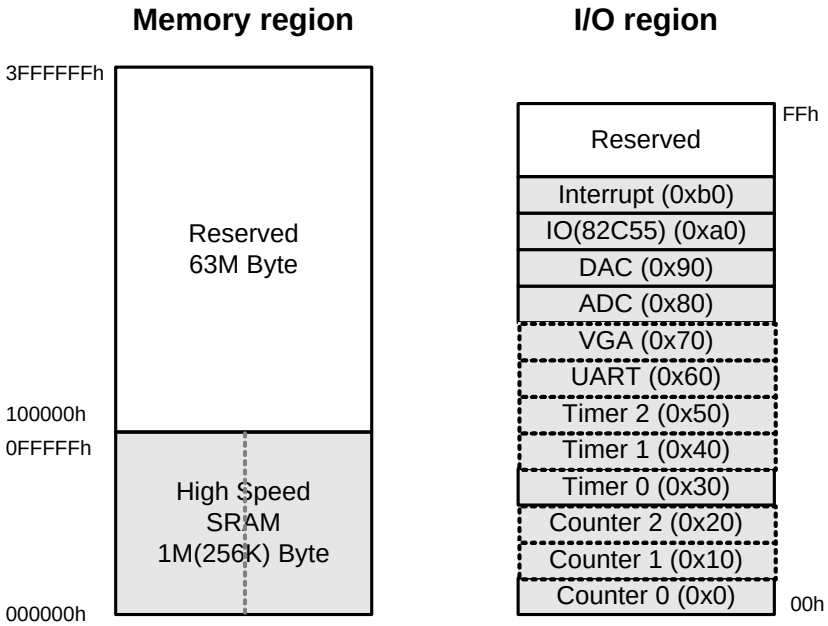
```
BOOL DeviceIoControl(
  HANDLE hDevice,                 // handle to device of interest
  DWORD dwIoControlCode,          // control code of operation to perform
  LPVOID lpInBuffer,              // pointer to buffer to supply input data
  DWORD nInBufferSize,            // size, in bytes, of input 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
);
```

위의 4개의 함수는 응용프로그램에서 하드웨어 장치를 제어하기 위하여 장치를 생성하고(소프트웨어의 관점), 데이터를 읽거나 쓰고 그리고 제어를 하기 위하여 마련된 것이다. PCI-EK01(A/B) 에서는 ReadFile 및 WriteFile API를 사용하지 않고 모든 제어 및 데이터 읽기/쓰기를 DeviceIoControl를 사용하여 구현하고 있다.

3. Hardware Resource

PCI-EK01에서 장치 기능의 제어는 메모리 및 I/O 영역에 위치하고 있다. 예를 들어 카운터 0의 제어를 하기 위하여는 I/O 영역 0번지의 제어 레지스터를 이용하면 된다.

PCI-EK01에서 메모리의 경우 총 64M byte, I/O의 경우 총 256Byte를 시스템에서 할당 받아 사용한다. 그림에서 보듯이 할당 받은 모든 영역을 사용하지는 않으며, 추후 기능 향상을 위하여 사용하지 않고 남겨둔 부분도 있다.



.

㈜ 1. I/O 장치 및 메모리에 대한 자세한 설명은 “AN203(PCI-EK01 Register Level Application Guide)”을 참조하기 바람.

㈜ 2. 상기 그림에서 사선으로 되어 있는 부분은 PCI-EK01(A)에서는 지원하지 않는 기능을 나타내고 있다 UART 및 VGA 는 추후 기능을 추가 할 예정임.

4. CreateFile

응용 프로그램에서 하드웨어 디바이스를 제어하기 위하여는 최초 CreateFile 함수를 이용하여 디바이스 파일을 생성하고 핸들을 얻어야 한다. 함수의 사용 예는 아래와 같다.

함수의 리턴 값으로 드라이버 핸들을 얻는다. 이 핸들 값은 추후 ReadFile, WriteFile, DeviceIoControl 함수에서 인수 값으로 사용된다.

```
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");
}
```

위에서 심볼릭 링크 이름으로 사용하는 인수 값(“\\\\.\\Pci_ek01_0”)은 시스템에 등록된 PCI-EK01 디바이스 장치의 이름이다.

시스템(PC)에 여러 개의 보드가 장착되어 있을 경우에는 “Pci_ek01_1” 같이 맨 뒷부분의 숫자가 증가되어 등록이 된다.

4. DeviceIoControl

CreateFile로 장치에 대한 디바이스 핸들을 얻게 되면, 이 핸들 값을 DiviceIoControl의 함수 인수로 상용하여 장치를 제어한다. 함수의 사용법은 아래와 같으며, 자세한 함수 설명은 MSDN을 참조 한다.

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

함수의 두 번째 인수로 사용되는 값은 “Control Code”값으로 이 값에 따라 다른 동작을 할 수 있으며, 사용하는 주요한 코드는 다음과 같다.

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

각 컨트롤 코드를 사용하여 DeviceIoControl 함수를 호출 할 때 bufInput에는 적절한 값이 있어야 하며, buf(Output Buffer)는 리턴 값을 저장할 충분한 공간을 확보하여야 한다.
버퍼 사이즈는 항상 바이트 단위로 지정한다.

Read 동작일 경우에는 첫 번째 인수 값으로 어드레스 Offset, 두 번째 인수로 읽을 바이트수가 32비트 DWORD 값으로 저장되며, 출력 buf는 읽을 데이터를 저장할 충분한 공간이 필요하다.
Write 동작일 경우에는 첫 번째 인수 값으로 어드레스 Offset, 두 번째 인수로 기록할 바이트수가 32비트 DWORD 값으로 저장되며 다음으로 기록할 데이터 값이 된다.

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