SHA – Sybera Hardware Access

Was ist SHA ?

Hardwarezugriffe unter Windows NT sind generell nur in der Kernel Ebene möglich. Um auf die Hardware zugreifen zu können ist es daher notwendig, einen s.g. Device Driver zu erstellen. Die Programmierung eines Device Drivers ist jedoch sehr aufwendig und benötigt ein tiefgreifendes Wissen über die Kernel Programmierung. Wir von SYBERA haben mit SHA ein Toolkit entwickelt, welches dem Programmierer den Zugriff auf die Hardware direkt von der Applikationsebene ermöglicht.

Das SHA Toolkit besteht aus vier Modulen:

- IO-PORT Zugriffe, Mapped Memory Zugriffe, PCI Bus Scan
- DMA Steuerung (Busmaster- und System-DMA)
- Interrupt-Steuerung (RING0 EXECUTION, FAST CALLBACK, FAST EVENT)
- Timer-Steuerung (RING0 EXECUTION, FAST CALLBACK, FAST EVENT)

IO-PORT Zugriffe, Mapped Memory Zugriffe, PCI Bus Scan

IO-Port Zugriffe und Zugriffe auf Mapped Memory sind i.d.R. die häufigsten Hardwarezugriffe. Das SHA Toolkit bietet hierbei die Möglichkeit die genutzten Hardware-Ressourcen in der Registry anzumelden oder auf bereits angemeldete Hardware-Ressourcen zu zugreifen. Zusätzlich bietet das SHA Toolkit eine Möglichkeit den PCI Bus zu scannen. Folgende Funktionen stehen zur Verfügung:

ShaRegisterPort

Exclusives Anmelden und Reservieren von IO-Ports (Falls diese nicht schon reserviert wurden). Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().Die einsetzbaren Interface-Typen sind in der Header-Datei „GLOBDEF.H“ definiert.

```
ULONG ShaRegisterPort(
            ULONG IfType,           //IN  : Interface type
            ULONG PA,               //IN  : Physical port address
            ULONG Size,             //IN  : Port range
            HANDLE* phPort)         //OUT : Handle to port device
```

ShaUnregisterPort

Freigabe der reservierten IO-Ports. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaUnregisterPort(HANDLE hPort)     //IN : Handle to port device
```

Beispiel:

```
HANDLE hPort;
DWORD Error = ShaRegisterPort(
                        IFT_PCIBUS,       //Interface type
                        0xE000,           //Physical IO base address
                        0x100,            //Port range
```

```
                        &hPort);          //Port handle
if (ERROR_SUCCESS == Error)
{
      //function succeeded
      ...
      Error = ShaUnregisterPort(hPort);
}
```

ShaReadPort

Lesen eines Zeichens vom IO-Port. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaReadPort(
            USHORT PortAddr,        //IN  : Physical port address
            PUCHAR pPortValue)      //OUT : Port value
```

Beispiel:

```
UCHAR c;
DWORD Error = ShaReadPort(0xE000, &c);
```

ShaWritePort

Schreiben eines Zeichens zum IO-Port. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaWritePort(
                  USHORT PortAddr,   //IN : Physical port address
                  UCHAR  PortValue)  //IN : Port value
```

Beispiel:

```
DWORD Error = ShaReadPort(0xE000, 0x55);
```

ShaMapMem

Reservierung eines mapped memory Speicherbereichs. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS und es wird ein gültiger Pointer „pUserVA“ auf den Speicherbereich zurückgegeben, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaMapMem(
            ULONG IfType,     //IN  : Interface type
            ULONG PA,         //IN  : Physical memory address
            ULONG Size,       //IN  : Memory range
            PVOID* ppUserVA,  //OUT : Pointer to user memory address space
            HANDLE* phMem)    //OUT : Referenced handle to memory device
```

ShaUnmapMem

Freigabe des mapped memory Speicherbereichs. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaUnmapMem(HANDLE hMem)      //IN : Handle to memory device
```

Beispiel:

```
HANDLE hMem;
PUCHAR pPort;
DWORD Error = ShaMapMem(
                  IFT_PCIBUS,       //Interface type
                  0xdc000000,       //Physical memory mapped base address
                  0x100,            //Memory range
                  (void**)&pPort,   //Referenced pointer to port memory
                  &hMem);           //Referenced port handle
if (ERROR_SUCCESS == Error)
{
      //Tauschen der Inhalte zweier Speicherbereiche
      c = pPort[0];
      pPort[0] = pPort[1];
      pPort[1] = c;
}
```

ShaScanPciBus

Auffinden der abstrahierten Basis-Adressen und des zugehörigen Hardware-Interrupts vom PCI-Bios. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS und die Basis-Adressen enthalten (abhängig von der Hardware) IO-Port, Mapped-Memory und ROM-Basis-Adressen, sowie die Interrupt-Nummer, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaScanPciBus(
            USHORT VendorID,              //IN  : Vendor ID
            USHORT DeviceID,              //IN  : Device ID
            ULONG BaseAddresses[6],       //OUT : PCI base addresses
            PUCHAR pInterruptLine)        //OUT : PCI interrupt line
```

Anmerkung:
Die ermittelten Addressen und Interrupt-Nummer sind vom System abstrahierte Werte, welche für den Einsatz in der virtuellen Windows-Umgebung bestimmt sind.

Beispiel:

```
USHORT IoBaseAddr;
ULONG BaseAddresses[6];
UCHAR InterruptLine;
DWORD Error = ShaScanPciBus(
                        0x1000,           //Vendor ID
                        0x0001,           //Device ID
                        BaseAddresses,    //PCI base addresses
                        &InterruptLine);  //PCI interrupt line
if (Error == ERROR_SUCCESS)
{
      for (int i=0; i<6; i++)
      {
            //Check for IO port address
            if (BaseAddresses[i] & (~1))
            {
                  IoBaseAddr = (USHORT)( BaseAddresses[0] & (~1));
                  ShaWritePort(IoBaseAddr, 0x55);
            }
      }
}
```

DMA Steuerung (Busmaster- und System-DMA)

Das SHA Toolkit bietet eine komfortable Möglichkeit sowohl Busmaster- als auch System-DMA Transfers auf Applikationsebene zu programmieren. Zudem können physikalisch zusammenhängende Speicherbereiche dynamisch angefordert und freigegeben werden.

ShaAllocMem

Anfordern und Reservieren von zusammenhägendem Physikalischem Speicher. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS und der Pointer pUserVA zeigt auf eine gültige virtuelle Addresse, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaAllocMem(
            ULONG IfType,         //IN     : Interface type
            BOOLEAN MasterFlag,   //IN     : Set system or busmaster DMA
            ULONG Channel,        //IN     : Set channel for system DMA
            ULONG* pPA,           //OUT    : Physical DMA address
            long* pSize,          //IN/OUT : DMA memory range
            PVOID* ppUserVA,      //OUT  : Pointer to DMA memory in user space
            HANDLE* phDma)        //OUT    : Handle to DMA device
```

Anmerkung:
Ob der angeforderte Speicher reserviert werden kann hängt systembedingt von den von Windows bereits verwalteten Resourcen und der damit verbundenen Fragmentierung des virtuellen Speichers ab. In der Regel lassen sich bis ca. 256KB zusammenhängender physikalischer Speicher reservieren.

ShaFreeMem

Freigabe des reservierten physikalischen Speichers. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaFreeMem(HANDLE hDma)       //IN : Handle to DMA device
```

Beispiel:

```
ULONG DmaBasePA;
ULONG DmaSize = 0x20000;
PUCHAR pDma;
HANDLE hDma;
DWORD Error = ShaAllocMem(
            IFT_PCIBUS,             //Interface type
            TRUE,                   //Set busmaster DMA
            0,                      //Required channel for system DMA
            &DmaBasePA,             //Physical address
            &DmaSize,               //Memory range
            (void**)&pDma,          //Pointer to user DMA memory
            &hDma);                 //Handle to DMA device
if (ERROR_SUCCESS == Error)
{
      for (int i=0; i<20000; i++)
            pDma[i] = (UCHAR)i;
}
```

ShaAllocChannel

Soll die systemeigene DMA-Hardware (DMA chip im PC) für den DMA-Transfer benutzt werden muß ein DMA-Channel vom System angefordert werden. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaAllocChannel(
                  BOOLEAN WriteFlag,      //IN : DMA direction
                  HANDLE hDma)            //IN : Handle to DMA device
```

Anmerkung:
der DMA Transfer ist unidirektional, d. h. es muß die Richtung für das Schreiben oder Lesen vom Hardware-Speicher in den System-Speicher angegeben werden.

Beispiel:

```
DWORD Error = ShaAllocChannel(TRUE, hDma);
```

Interrupt-Steuerung

Bei der Entwicklung der Interrupt-Steuerung von SHA wurde größten Wert auf bestmögliche Zyklus- und Latenzzeiten gelegt. Das SHA-Toolkit bietet dem Programmierer die Möglichkeit, spezielle Routinen direkt auf Applikations-Ebene zu definieren und diese im Ring0 (Kernel-Ebene) ausführen zulassen. Die somit erzielten Latenzzeiten (z.B. bei der Interrupt-Steuerung) unterscheiden sich nicht von den Latenzzeiten der Treiber Programmierung.
Je nach Komplexität und Priorität der auszuführenden Interrupt- oder Timer-Steuerung bietet das SHA-Toolkit 3 unterschiedliche Programmiertechniken um auf Hardware Ereignisse zu reagieren.

ShaConnectInterrupt

Anmelden des Interrupts an das System und Festlegen der Interrupt-Steuerung. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS und der String „szEventName“ enthält den Event-Namen, der zum Synchronisieren des Interrupt-Ereignisses benutzt werden kann, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaConnectInterrupt(
            ULONG IfType,                 //IN  : Interface type
            FP_RING0 fpUserRing0,         //IN  : Optional RING0 EXECUTION
                                          //      function, otherwise NULL
            FP_CALLBACK fpUserCallback,   //IN  : Optional FAST CALLBACK
                                          //      function, otherwise NULL
            HANDLE hDma,                  //IN  : Optional handle to DMA device,
                                          //      otherwise NULL
            UCHAR InterruptLine,          //IN  : Interrupt number
            char* szEventName,            //OUT : Interrupt event name
            HANDLE* phIntr)               //OUT : Handle to interrupt device
```

Anmerkung:
die unterschiedlichen Programmier-Techniken für die Interrupt-Steuerung können beliebig kombiniert werden. Falls bei Interrupt Ereignissen ein DMA Transfer ausgeführt werden muß, muß das entsprechende HANDLE des zugehörigen DMA Device mit angegeben werden.

ShaDisconnectInterrupt

Freigabe des Interrupts unter Interrupt-Steuerung. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaDisconnectInterrupt(HANDLE hIntr)     //IN : Handle to interrupt device
```

Beispiel:

```
char szEventName[EVENT_NAME_SIZE];
ULONG BaseAddresses[6];
UCHAR InterruptLine;
DWORD Error = ShaScanPciBus(
                        0x1000,          //Vendor ID
                        0x0001,          //Device ID
                        BaseAddresses,   //PCI base addresses
                        &InterruptLine); //PCI interrupt line
if (Error == ERROR_SUCCESS)
{
      //Set interrupt for busmaster DMA
      Error = ShaConnectInterrupt(
                  IFT_PCIBUS,       //Interface type
                  NULL,             //Optional RING0 EXECUTION routine
                  NULL,             //Optional FAST CALLBACK routine
                  NULL,             //Optional handle to DMA device (only
                                    //required for slave DMA)
                  InterruptLine,    //Interrupt number
                  szEventName,      //Interrupt event name
                  &hIntr);          //Interrupt handle

      if (Error == ERROR_SUCCESS)
      {
            //Open interrupt event with the received
            //event name by ShaConnectInterrupt
            HANDLE hEvent = OpenEvent(SYNCHRONIZE, FALSE, szEventName);
            if (hEvent)
            {
                  //Wait for interrupt event
                  WaitForSingleObject(hEvent, INFINITE);
                  ...
                  CloseHandle(hEvent);
            }
            Error = ShaDisconnectInterrupt(hIntr);
      }
}
```

RING0 EXECUTION

Die RING0 EXECUTION Methode arbeitet mit der höchsten Priorität, welche für Kernel-Routinen zur Verfügung steht und kann sowohl für die Interrupt- als auch für die Timer-Steuerung eingesetzt werden. Die RING0 EXECUTION Routine dient i.d.R. zur Rücksetzung des Interrupt-Controllers, sowie kleinerer Datentransfers und ist mit gewissen Einschränkungen verbunden. So können in der RING0 EXECUTION Routine nur Systembefehle (z.B. SHA_SYSTEM_READPORT / SHA_SYSTEM _WRITEPORT) und Standard Operationen ausgeführt werden. Es entfällt die Möglichkeit innerhalb der RING0 EXECUTION Routine Breakpoints zu setzen.

Beispiel:

```
//System data declaration for use in ring0
```

```
#pragma data_seg(".sdata")
      UCHAR Value;
#pragma data_seg()

//System code declaration for use in ring0
#pragma code_seg(".scode")
void static Ring0Routine(void)
{
      SHA_SYSTEM_WRITEPORT(0xE000, 0x55);
      SHA_SYSTEM_READPORT(0xE000, &Value);
}
#pragma code_seg()


//Set interrupt for busmaster DMA
Error = ShaConnectInterrupt(
                  IFT_PCIBUS,       //Interface type
                  Ring0Routine,     //Optional RING0 EXECUTION routine
                  NULL,             //Optional FAST CALLBACK routine
                  NULL,             //Optional handle to DMA device (only
                                    //required for slave DMA)
                  InterruptLine,    //Interrupt number
                  szEventName,      //Interrupt event name
                  &hIntr);          //Interrupt handle

if (Error == ERROR_SUCCESS)
{
      while (Value != 0x55);
      ...
}
```

FAST CALLBACK

Die FAST CALLBACK Methode arbeitet mit einer niedrigeren Priorität als die RING0 EXECUTION. Die FAST CALLBACK Methode ermöglicht jedoch dem Programmierer WIN32API Funktionen sowohl in der Interrupt-Steuerung, als auch in der Timer -Steuerung einzusetzen. Zudem können Breakpoints innerhalb der Routine gesetzt werden.

Anmerkung:
Die FAST CALLBACK Methode eignet sich jedoch nur für nicht-kontinuierliche Aufrufe, da sie asynchron arbeitet.

Beispiel:

```
void CallbackRoutine(PVOID, PVOID, PVOID)
{
     MessageBox(NULL, "FAST Callback OK !", "SHA Info",
MB_OK);
}

//Set interrupt for busmaster DMA
Error = ShaConnectInterrupt(
                  IFT_PCIBUS,       //Interface type
                  NULL,             //Optional RING0 EXECUTION routine
                  CallbackRoutine,  //Optional FAST CALLBACK routine
                  NULL,             //Optional handle to DMA device (only
                                    //required for slave DMA)
                  InterruptLine,    //Interrupt number
                  szEventName,      //Interrupt event name
                  &hIntr);          //Interrupt handle
```

```
if (Error == ERROR_SUCCESS)
{
      ...
}
```

FAST EVENT

die FAST EVENT Methode ermöglicht es dem Programmierer eine Applikation oder ein Thread mit einem Interrupt- oder Timer-Ereignis zu synchronisieren.

Alle 3 Methoden können beliebig kombiniert werden.

Beispiel:

```
//Set interrupt for busmaster DMA
Error = ShaConnectInterrupt(
                  IFT_PCIBUS,       //Interface type
                  Ring0Routine,     //Optional RING0 EXECUTION routine
                  CallbackRoutine,  //Optional FAST CALLBACK routine
                  NULL,             //Optional handle to DMA device (only
                                    //required for slave DMA)
                  InterruptLine,    //Interrupt number
                  szEventName,      //Interrupt event name
                  &hIntr);          //Interrupt handle

if (Error == ERROR_SUCCESS)
{
      //Open interrupt event with the received
      //event name by ShaConnectInterrupt
      HANDLE hEvent = OpenEvent(SYNCHRONIZE, FALSE, szEventName);
      if (hEvent)
      {
            //Wait for interrupt event
            WaitForSingleObject(hEvent, INFINITE);
            ...
            CloseHandle(hEvent);
      }
}
```

Timer-Steuerung

Das Timer Modul ermöglicht die Steuerung bis auf ca. 1ms. Dadurch können genaue TimeOut-Steuerungen oder Zeitmessungen vorgenommen werden. Wie bei der Interrupt-Steuerung können RING0 EXECUTION, FAST CALLBACK und FAST EVENT in beliebiger Kombinationen benutzt werden. Das Timer Modul bietet die Möglichkeit einen periodischen Timer oder einen SINGLE SHOT Timer zu programmieren.

ShaConnectTimer

Anmelden eines Timers an das System und Festlegen der Timer-Steuerung. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS und der String „szEventName“ enthält den Event-Namen, der zum Synchronisieren des Timer-Ereignisses benutzt werden kann, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaConnectTimer(
            long        DueTime,          //IN  : Timer due time in msec
            long        Period,           //IN  : Timer period in msec
```

```
            FP_RING0    fpUserRing0,      //IN  : Optional RING0 EXECUTION
                                          //      function, otherwise NULL
            FP_CALLBACK fpUserCallback,   //IN  : Optional FAST CALLBACK
                                          //      function, otherwise NULL
            char*       szEventName,      //OUT : Timer event name
            HANDLE*     phTimer)          //OUT : Handle to timer device
```

Anmerkung:
die unterschiedlichen Programmier-Techniken für die Timer-Steuerung können beliebig kombiniert werden.

ShaDisconnectTimer

Abmelden des Timers vom System. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaDisconnectTimer(HANDLE hTimer)        //IN : Handle to timer device
```

ShaStartTimer

Sofortiges Aktivieren des Timers. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaStartTimer(HANDLE hTimer)             //IN : Handle to timer device
```

ShaStopTimer

Sofortiges Anhalten eines periodischen Timers. Bei Erfolg ist der Rückgabewert ERROR_SUCCESS, andernfalls entspricht der Rückgabewert dem von GetLastError().

```
ULONG ShaStopTimer(HANDLE hTimer)              //IN : Handle to timer device
```

Anmerkung:
diese Funktion dient Anhalten eines periodischen Timers. Bei SINGLE-SHOT Timern wird diese Funktion nicht benötigt.

Beispiel:

```
//System data declaration for use in ring0
#pragma data_seg(".sdata")
      ULONG Ring0Count = 0;
#pragma data_seg()

//System code declaration for use in ring0
#pragma code_seg(".scode")
void static Ring0Routine(void)
{
      //Increment our count
      Ring0Count++;
}
#pragma code_seg()


HANDLE hTimer;
DWORD Error;
char szEventName[EVENT_NAME_SIZE];

//Setup timer
```

```
Error = ShaConnectTimer(
                  0,                //Due time [msec]
                  100,              //Period [msec]
                  Ring0Routine,     //RING0 EXECUTION routine, optional
                  NULL,             //FAST CALLBACK routine, optional
                  szEventName,      //Timer event
                  &hTimer);         //Timer handle
if (Error == ERROR_SUCCESS)
{
      //Open timer event with the received
      //event name by ShaConnectInterrupt
      HANDLE hEvent = OpenEvent(SYNCHRONIZE, FALSE, szEventName) ;
      if (hEvent)
      {
            //To see, how our timer event works, wait for the first event
            WaitForSingleObject(hEvent, INFINITE);

            //Start the timer immediately
            ShaStartTimer(hTimer);

            //Stop the current thread (The RING0 Routine will
            //be called each 100ms
            Sleep(5000);

            //Stop the timer immediately
            ShaStopTimer(hTimer);
            //Close event
            CloseHandle(hEvent);
      }

      //Release timer
      Error = ShaDisconnectTimer(hTimer);
}
```