

DSP563xx HI32 PCI Functions

Ilan Naslavsky
Leonid Smolyansky

This document describes the DSP563xx_HI32_PCI framework, a set of functions in C that enable the user to operate any DSP56300 device with an HI32 interface (for example, DSP56301, DSP56305) through the PCI bus of a host running Microsoft® Windows® 95/98. The DSP563xx_HI32_PCI framework also includes a virtual device driver (VxD), DSPVXD.VXD, to support these functions. This document complements AN1780/D, *DSP563xx HI32 as A PCI Agent*, which describes the usage of the HI32 as a PCI.

All DSP563xx_HI32_PCI functions run in ring-3 of the operating system, assisted by the DSPVXD.VXD, which provides all required ring-0 services. Use of the functions does not require any ring-0 coding. The following functionality is provided:

- Read and write HI32 host-side registers
- Data blocks read from and write to DSP563xx
- Read and write HI32 configuration space
- Download DSP563xx code
- Interrupt support
- Bus mastering support functions
 - Linear-to-physical address conversion
 - Memory pages lock and unlock.

Appendix A presents the source code of all the functions. Appendix B presents the source code of the VxD, although use of the compiled VxD file is sufficient to use the functions. Also provided are:

- Object DSPVXD.VXD
- A running sample and its sources
- An DSP56301 assembly program that reacts to the initialization of the functions used in the sample

You can download a README file with preliminary information and a compressed ZIP-format file containing the files, DSP563xx_HI32_PCI.ZIP, at the following location:

<http://www.mot.com/SPS/DSP/Documentation/appnotes.html>

Contents

1	DSP563xx_HI32_PCI Framework.....	1-2
1.1	Application Files.....	1-2
1.2	DSP56301 Assembly Program	1-2
1.3	VxD Services	1-2
1.4	Development Environment	1-3
2	Functions	2-1
A	Function Code	A-1
A.1	DSP563xx_LoadVxD	A-1
A.2	DSP563xx_UnLoadVxD	A-2
A.3	DSP563xx_InitializeDevice	A-3
A.4	DSP563xx_ReadHI32Register	A-4
A.5	DSP563xx_WriteHI32Register	A-5
A.6	DSP563xx_ReadCfgSpace	A-6
A.7	DSP563xx_WriteCfgSpace	A-7
A.8	DSP563xx_DownloadCode	A-8
A.9	DSP563xx_WaitForInterrupt	A-10
A.10	DSP563xx_GetPhysAdd	A-11
A.11	DSP563xx_LockMemoryPage ...	A-12
A.12	DSP563xx_UnLockMemoryPage	A-13
A.13	DSP563xx_ChangeDataMode ...	A-14
A.14	Auxiliary Functions	A-15
A.15	DSP563xx_HI32_PCI Functions - C-Header file	A-16
B	Code Listing	B-1
B.1	Virtual Device Code - VxD Source - C-code	B-1
B.2	Virtual Device Code - VxD Source - C-Header file	B-8
B.3	DSP56301 Assembly Code	B-9
B.4	DSP56301 Sample - C-code	B-13
B.5	DSP56301 Sample - C-Header file	B-24

1 DSP563xx_HI32_PCI Framework

This section presents an overview of the DSP563xx_HI32_PCI framework.

1.1 Application Files

Accompanying this application note is a README file with preliminary directions and a compressed ZIP-format file, DSP563xx_HI32_PCI.ZIP, containing the following files:

DSPVXD.VXD	Microsoft® Windows® 95/98 virtual device driver for any board containing a DSP563xx with HI32
DSPVXD.C:	Source C-code of DSPVXD.VXD
DSPVXD.H	C-header file for DSPVXD.VXD
DSP56301.EXE	Microsoft® Windows® 95/98 console application provided as a sample of functions usage
DSP56301.C	Source C-code of DSP56301.EXE
DSP56301.H	C-header file for DSP56301.c
DSP56301.ASM	DSP56301 sample assembly code for interacting with the provided functions
DSP56301.PCI	ASCII file with DSP56301.ASM assembled code formatted for downloading through the PCI bus to the DSP56301

1.2 DSP56301 Assembly Program

The assembly code programs the DSP56301 to interact with the DSP563xx_HI32_PCI functions demonstrated in the DSP56301.EXE sample. The following routines are implemented:

- Host command ISRs for setting and resetting host flags, asserting/deasserting the INTA PCI line, and performing a personal reset on the HI32 for changing PCI slave data format
- DMA configuration for servicing HI32 slave FIFOs (both input and output, one DMA channel per direction, serving circular data buffers of size 32 words (24-bits)); output buffer is filled with general data
- HI32 personal reset and PCI mode initialization
- Checksum calculation

1.3 VxD Services

The DSPVXD.VXD VxD provides the following services to the DSP563xx_HI32_PCI function:

- Initialization: HI32 base address retrieval, interrupt hooking
- Memory locking: locking and unlocking memory pages
- PCI configuration space access: read and write the HI32 configuration space
- Interrupt handling
- Linear-to-physical address conversion

1.4 Development Environment

The software described in this document was developed in the following environment:

- VxD:
 - *C Compiler*: Microsoft Visual C++®, version 5.0
 - *Main Library*: Vireo Software VtoolsD, version 2.01
- DSP563xx_HI32_PCI functions:
 - *C Compiler*: Microsoft Visual C++, version 5.0
- DSP56301 Assembly:
 - *Environment*: Motorola DSP Development Environment

Note: Neither development environment item is necessary for running the sample provided. Also, all driver-related files, source code, executable files, and VxD type files are provided as-is as an example of implementation. They have not passed exhaustive verification and validation on varied PC platforms. It is your responsibility to resolve any Microsoft Windows 95/98 software-related problems. Motorola provides technical support only for issues directly related to the DSP56300 family.

2 Functions

This section describes every function in the DSP563xx_HI32_PCI framework and the *DSPINFO* data structure that characterizes the device accessed by the functions. Following is an overview listing of the functions that shows the page number for each function description.

DSPINFO	DSP563xx Device Descriptor	2
DSP563xx_LoadVxD	Load Virtual Device Driver (VxD)	3
DSP563xx_UnLoadVxD	Unload Virtual Device Driver (VxD)	4
DSP563xx_InitializeDevice	Retrieve Device Information from Configuration Manager	5
DSP563xx_ReadHI32Register	Read DSP563xx HI32 Host-side Register	6
DSP563xx_WriteHI32Register	Write DSP563xx HI32 Host-side Register	7
DSP563xx_ReadCfgSpace	Read a 32-bit Register in the HI32 Configuration Space.....	8
DSP563xx_WriteCfgSpace	Write a 32-bit Register in the HI32 Configuration Space.....	9
DSP563xx_DownloadCode	Download DSP563xx Code Through HI32	10
DSP563xx_WaitForInterrupt	Wait Until VxD Signals That the HI32 PCI Interrupt Occurred.....	11
DSP563xx_GetPhysAdd	Convert Linear Address to Physical Address	12
DSP563xx_LockMemoryPage	LockLinear Memory Pages of a Buffer	13
DSP563xx_UnLockMemoryPage	Unlock Linear Memory Pages of a Buffer	14
DSP563xx_ChangeDataMode	Change PCI Slave Data Format	15

DSPINFO

DSP563xx Device Descriptor

Syntax

```
typedef struct
    char* VxDFileName;
    HANDLE DeviceHandle;
    HANDLE Evnt0;
    HANDLE Evnt3;
    DWORD DeviceId;
    DWORD HI32BaseAddress;

} DSPINFO , *pDSPINFO;
```

Parameters

<i>VxDFileName</i>	Name of Virtual Device Driver file
<i>DeviceHandle</i>	Handle to the Virtual Device Driver
<i>Evnt0</i>	Handle to a ring-0 event common to ring-0 and ring-3
<i>Evnt3</i>	Handle to a ring-3 event common to ring-0 and ring-3
<i>DeviceId</i>	Device ID number of the used DSP563xx
<i>HI32BaseAddress</i>	Memory Base Address of DSP563xx

Description This data structure contains all the information about the specific device that is needed by the functions in the DSP563xx_PCI_FUNCTIONS framework. You must explicitly fill part of this structure. The remaining parameters are filled by specific function calls:

1. *VxDFileName*, *DeviceId* must be filled by the user.
2. *DeviceHandle*, *Evnt0*, *Evnt3* are filled by calling **DSP563xx_LoadVxD**.
3. *HI32BaseAddress* is filled by calling **DSP563xx_InitializeDevice**.

Example

```
// Device Declaration
pDSPINFO DSP56301;
DSP56301->DeviceId = 0x1801;
DSP56301->VxDFileName = "\\.\DSPVXD.VXD";

// this call fills up the values of DeviceHandle, Evnt0 and Evnt3
DSP563xx_LoadVxD(DSP56301);

// this call fills up the value of HI32BaseAddress
DSP563xx_InitializeDevice(DSP56301);
```

See Also

DSP563xx_InitializeDevice, **DSP563xx_LoadVxD**

DSP563xx_LoadVxD

Load Virtual Device Driver (VxD)

Syntax

```
BOOL DSP563xx_LoadVxD(pDSPINFO DSP563xx)
```

Parameters

DSP563xx Pointer to a DSP563xx device descriptor

Description Performs three actions:

1. Creates an event common to ring-3 and ring-0 and initializes *DSP563xx->Evtnt0* and *DSP563xx->Evtnt0*.
2. Loads the virtual device driver defined by *DSP563xx->VxDFileName*.
3. Initializes *DSP563xx->DeviceHandle* with the handle to the loaded VxD.

Returns

TRUE if a common event is created and the VxD successfully loads. FALSE otherwise.

Example

```
BOOL bReturnValue;  
  
// ... Device Declaration ...  
  
printf("Loading %s : ", DSP56301->VxDFileName);  
bReturnValue = DSP563xx_LoadVxD(DSP56301);  
if (bReturnValue){  
    printf(" Passed\n");  
}  
else{  
    printf(" Failed\n");  
}
```

See Also

DSP563xx Device Descriptor, **DSP563xx_InitializeDevice**, **DSP563xx_UnLoadVxD**

DSP563xx_UnLoadVxD

Unload Virtual Device Driver (VxD)

Syntax

```
BOOL DSP563xx_UnLoadVxD(pDSPINFO DSP563xx)
```

Parameters

DSP563xx Pointer to a DSP563xx device descriptor

Description Unloads the virtual device driver defined by *DSP563xx->VxDFileName*.

Returns

TRUE if the VxD unloads. FALSE otherwise.

Example

```
BOOL     bReturnValue;  
  
// Device Declaration, call to DSP563xx_LoadVxD and Device Initialization ...  
  
printf("Unloading VxD:");  
bReturnValue = DSP563xx_UnLoadVxD(DSP56301);  
if (bReturnValue)  
{  
    printf(" Passed\n");  
}  
else  
{  
    printf(" Failed\n");  
}
```

See Also

DSP563xx Device Descriptor, **DSP563xx_InitializeDevice**, **DSP563xx_LoadVxD**

DSP563xx_InitializeDevice

Retrieve Device Information From Configuration Manager

Syntax

```
BOOL DSP563xx_InitializeDevice(pDSPINFO DSP563xx)
```

Parameters

DSP563xx Pointer to a DSP563xx device descriptor

Description Accesses the configuration manager and performs the following actions:

1. Enables the device's interrupt service.
2. Locks a linear memory page containing the HI32 memory base address.
3. Fills the DSP563xx device descriptor with the device's memory base address.

Returns

TRUE if the initialization process succeeds. FALSE if the initialization process fails (could not find the referenced DSP563xx device).

Example

```
BOOL     bReturnValue;  
  
// Device Declaration, call to DSP563xx_LoadVxD  
  
   bReturnValue = DSP563xx_InitializeDevice(DSP56301);  
   if (bReturnValue){  
       printf("HI32 Base Address :%8lx\n",DSP56301->HI32BaseAddress);  
   }  
   else{  
       printf("Device Node Not Found\n");  
   }  
}
```

See Also

DSP563xx Device Descriptor, **DSP563xx_UnLoadVxD**, **DSP563xx_LoadVxD**

DSP563xx_ReadHI32Register

Read DSP563xx HI32 Host-side Register

Syntax

```
DWORD DSP563xx_ReadHI32Register(pDSPINFO DSP563xx , DWORD Register)
```

Parameters

DSP563xx Pointer to a DSP563xx device descriptor
Register Register Offset

Description Reads an HI32 host-side register.

Returns

The contents of the register.

Example

```
#define HCTR_OFFSET            0x00000004

// ... Initialization ...

printf ("    HCTR : %8lx\n", DSP563xx_ReadHI32Register(DSP56301 , HCTR_OFFSET));
```

See Also

DSP563xx_WriteHI32Register

DSP563xx_WriteHI32Register

Write DSP563xx HI32 Host-side Register

Syntax

```
void DSP563xx_WriteHI32Register(pDSPINFO DSP563xx , DWORD Register, DWORD  
NewValue)
```

Parameters

<i>DSP563xx</i>	Pointer to a DSP563xx device descriptor
<i>Register</i>	Register Offset
<i>NewValue</i>	Value to be written to register

Description Writes a given value to an HI32 host-side register.

Returned Value

This function does not return any value.

Example

```
#define HCTR_OFFSET          0x00000004  
  
// ... Initialization ...  
  
//Sending Host Command 0xEC  
DSP563xx_WriteHI32Register(DSP56301, HCTR_OFFSET, 0xED);
```

See Also

DSP563xx_ReadHI32Register, DSP563xx_ReadData, DSP563xx_WriteData

DSP563xx_ReadCfgSpace

Read a 32-bit Register in the HI32 Configuration Space

Syntax

```
BOOL DSP563xx_ReadCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD* CfgWord)
```

Parameters

<i>DSP563xx</i>	Pointer to a DSP563xx device descriptor
<i>CfgOffset</i>	Offset of the 32-bit register in the HI32 Configuration Space
<i>CfgWord</i>	Pointer to a buffer for receiving the read double-word

Description Directs the VxD to read a 32-bit register in the HI32 configuration space, located in the provided offset.

Returns

TRUE if the read of configuration space succeeds; FALSE if the read of the configuration space fails.

Example

```
BOOL bReturnValue;  
DWORD dwReturnValue;  
  
// ... Initialization ...  
  
printf("Reading CSTR/CCMR Registers\n");  
bReturnValue = DSP563xx_ReadCfgSpace(0x4, DSP56301, &dwReturnValue );  
if (bReturnValue){  
    printf("Read Configuration Space : %8lx\n",dwReturnValue);  
}  
else{  
    printf("Failed to Read Configuration Space\n");  
}
```

See Also

DSP563xx_WriteHI32Register, DSP563xx_ReadHI32Register, DSP563xx_WriteCfgSpace, DSP563xx_WriteData, DSP563xx_ReadData

DSP563xx_WriteCfgSpace

Write a 32-bit Register in the HI32 Configuration Space

Syntax

```
BOOL DSP563xx_WriteCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD* CfgWord)
```

Parameters

<i>DSP563xx</i>	Pointer to a DSP563xx device descriptor
<i>CfgOffset</i>	Offset of the 32-bit register in the HI32 configuration space
<i>CfgWord</i>	Pointer to a buffer containing the double-word to be written

Description Directs the VxD to write a 32-bit register in the HI32 configuration space, which is located in the offset provided.

Returns

TRUE if the write to configuration space succeeds; FALSE if the write to configuration space fails.

Example

```

BOOL bReturnValue;
DWORD dwReturnValue;
DWORD dwNewValue;

// ... Initialization ...

// Read CSTR/CCMR Register (offset = $4), then write it
bReturnValue = DSP563xx_ReadCfgSpace(0x04, DSP56301, &dwReturnValue );
dwNewValue = dwReturnValue | 0x00000004; // set BM bit
bReturnValue = DSP563xx_WriteCfgSpace(0x04, DSP56301, dwNewValue );
if (bReturnValue){
    printf("Written Configuration Space\n");
}
else{
    printf("Failed to Write Configuration Space\n");
}

```

See Also

DSP563xx_WriteHI32Register, DSP563xx_ReadHI32Register, DSP563xx_ReadCfgSpace, DSP563xx_WriteData, DSP563xx_ReadData

DSP563xx_DownloadCode

Download DSP563xx Code Through HI32

Syntax

```
BOOL DSP563xx_DownloadCode(pDSPINFO DSP563xx , const CHAR *Filename);
```

Parameters

DSP563xx Pointer to a DSP563xx device descriptor
Filename Name of the file containing the code to be downloaded (in *.pci format)

Description Reads the file *Filename* and writes its contents to the *DSP563xx*, which should be running the chip's bootstrap program, in Host Bootstrap PCI Chip Operation mode. Section 2 of AN1780/D, entitled *DSP563xx HI32 As A PCI Agent*, recommends a complete bootstrapping procedure and describes the *.pci format.

Returns

TRUE if the checksum received from the DSP563xx equals the checksum computed by the function; FALSE otherwise.

Example

```
BOOL bReturnValue;  
  
// ... Initialization ...  
  
bReturnValue = DSP563xx_DownloadCode(DSP56301, "dsp56301.pci");  
if (bReturnValue) {  
    printf("Succeeded to Download Code\n");  
}  
else {  
    printf("Failed to Download Code\n");  
}
```

See Also

[DSP563xx_ReadData](#), [DSP563xx_WriteData](#), [DSP563xx_WriteHI32Register](#), [DSP563xx_ReadHI32Register](#)

DSP563xx_WaitForInterrupt

Wait Until VxD Signals That the HI32 PCI Interrupt Occurred

Syntax

```
DWORD DSP563xx_WaitForInterrupt(pDSPINFO DSP563xx , DWORD Timeout)
```

Parameters

DSP563xx Pointer to a DSP563xx device descriptor
Timeout Time (in milliseconds) to wait.

Description Waits until one of the following two situations occurs:

1. In case the HI32 PCI interrupt occurs during this call, the VxD handles the interrupt and notifies the application of the interrupt occurrence by setting the event created by **DSP563xx_LoadVxD** (common to ring-0 and ring-3).
2. *Timeout* milliseconds have passed. *Timeout* must be set to INFINITE if the function should return only when the event is set (i.e., until the interrupt occurs).

This function enables the DSP563xx to index its PCI interrupt with eight possible values by writing the HI32 host flags (HF5-HF3 in HSTR) before asserting the interrupt line. This function reads the host flags to identify which indexed interrupt has occurred.

Returns

WAIT_TIMEOUT in case *Timeout* milliseconds pass and the event is not set. If *Timeout* is INFINITE, the function does not return until the event is set. Otherwise, the function returns the index of the interrupt (i.e., the number between zero and seven coded on the host flags).

Example

```
DWORD dwReturnValue;
// ... Initialization ...
printf ("Waiting for notification from VxD\n");
dwReturnValue = DSP563xx_WaitForInterrupt(DSP56301, 1000);
if (dwReturnValue == WAIT_TIMEOUT) {
    printf ("Interrupted Timed Out\n");
}
else {
    printf ("Received Interrupt %8lx\n", dwReturnValue);
}
```

See Also

DSP563xx_IntAck, DSP563xx_InitializeDevice, DSP563xx_WriteHI32Register

DSP563xx_GetPhysAdd

Convert Linear Address to Physical Address

Syntax

```
DWORD DSP563xx_GetPhysAdd(DWORD LinAd, pDSPINFO DSP563xx)
```

Parameters

DSP563xx Pointer to a DSP563xx device descriptor
LinAd Linear address to be converted

Description Calls the VxD to convert a given linear address to a physical address. The physical address is the one that can be referred to in PCI transactions. The physical address retrieved with this function is useful only if the linear memory page matching the provided linear address is locked with a call to **DSP563xx_LockMemoryPage**. If the page is not locked, the operating system can change the relation physical-to-linear any time.

Returns

The physical address corresponding to the given linear address.

Example

```
// The initial call to DSP563xx_InitializeDevice makes the VxD lock a linear
// page and commit it to the HI32 Memory Base Address. The linear address
// available in the DSP56301 device descriptor is used in this example.
// The user can check the validity of the physical address by checking
// the System Manager utility

printf("Getting the physical memory base address of HI32:");
printf ("%08lx\n", DSP563xx_GetPhysAdd(DSP56301->HI32BaseAddress,
                                       DSP56301));
```

See Also

DSP563xx_LockMemoryPage, **DSP563xx_UnLockMemoryPage**, **DSP563xx_InitializeDevice**

DSP563xx_LockMemoryPage

Lock Linear Memory Pages of a Buffer

Syntax

```
BOOL DSP563xx_LockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size)
```

Parameters

<i>DSP563xx</i>	Pointer to a DSP563xx device descriptor
<i>LinearAddress</i>	Pointer to a buffer in which linear memory pages must be locked
<i>Size</i>	Size of the buffer, in bytes

Description Calls the VxD to lock the linear memory pages corresponding to a buffer in the linear addresses scope.

Returns

TRUE if the memory pages successfully lock; FALSE if the memory pages fail to lock.

Example

```
DWORD* LinearAddress;
// ... Initialization ...

LinearAddress = (DWORD*)malloc(0x400*sizeof(DWORD));
// locks a 1k-Dwords buffer
if(DSP563xx_LockMemoryPage(LinearAddress,DSP56301,
                          0x400*sizeof(DWORD))) {
    printf("Buffer Pages Locked\n");
}
else {
    printf("Failed to Lock Buffer Pages\n");
}
```

NOTE 1: The number of locked pages is calculated with the complete address, including its offset. Thus, if a one-page buffer is locked with **DSP563xx_LockMemoryPage** from a linear address with an offset that is not zero, two pages are actually locked. To lock the exact number of pages, the linear address given to the **DSP563xx_LockMemoryPage** function should be aligned with the beginning of the page.

NOTE 2: The **DSP563xx_LockMemoryPage** function does not constrain the locked pages to be contiguous. When the physical address is to be given to the DSP563xx for PCI bus mastering transactions, the burst length must take the address offset into account so that the access does not fall out of a locked page.

See Also

DSP563xx_UnLockMemoryPage, **DSP563xx_GetPhysAdd**

DSP563xx_UnLockMemoryPage

Unlock Linear Memory Pages of a Buffer

Syntax

```
BOOL DSP563xx_UnLockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD Size)
```

Parameters

<i>DSP563xx</i>	Pointer to a DSP563xx device descriptor
<i>LinearAddress</i>	Pointer to a buffer in which linear memory pages must be unlocked
<i>Size</i>	Size of the buffer, in bytes

Description Calls the VxD to unlock the linear memory pages corresponding to the buffer in the linear addresses scope, previously locked with **DSP563xx_LockMemoryPage**.

Returns

TRUE if the memory pages successfully unlock; FALSE if the memory pages fail to unlock.

Example

```
DWORD* LinearAddress;  
// ... Initialization ...  
    LinearAddress = (DWORD*)malloc(0x400*sizeof(DWORD));  
// locks a 1k-Dwords buffer  
// ...  
// unlocks it  
    if(DSP563xx_UnLockMemoryPage(LinearAddress, DSP56301,  
                                0x400*sizeof(DWORD))) {  
        printf("Buffer Pages Unlocked\n");  
    }  
    else {  
        printf("Failed to Unlock Buffer Pages\n");  
    }  
}
```

See Also

DSP563xx_LockMemoryPage, **DSP563xx_GetPhysAdd**

DSP563xx_ChangeDataMode

Change PCI Slave Data Format

Syntax:

```
void DSP563xx_ChangeDataMode(pDSPINFO DSP563xx , DWORD HI32DataMode)
```

Parameters:

DSP563xx	-Pointer to a DSP563xx device descriptor
HI32DataMode	- Data Mode: HI32_24BIT_MODE (HRF = HTF = \$1) = 0x0040000 HI32_32BIT_MODE (HRF = HTF = \$0) = 0x0000000

Description: This function changes the PCI Slave Data Format for the HI32 by setting the bits HTF[1:0] and HRF[1:0] in HCTR. The function sends a host command whose Interrupt Service Routine (ISR) performs the following actions:

1. Put the HI32 in mode 0 (bits HM[1:0]=0)
2. Wait until HACT bit is cleared (personal reset)
3. Put the HI32 back in PCI mode (bits HM[1:0]=1)
4. Interrupts PC to acknowledge ISR
- 5.

Returns:

This function does not return any value.

Example:

```
// ... Device Declaration ...

// Change HI32 PCI Slave Data Format to HI32_24BIT_MODE
    DSP563xx_ChangeDataMode(DSP56301 , HI32_24BIT_MODE);
```

See Also:

DSP563xx Device Descriptor, **DSP563xx_InitializeDevice**, **DSP563xx_DownloadCode**

A Function Code

This appendix presents the source code for every function in the DSP563xx_HI32_PCI framework and the C header file required by the functions.

A.1 DSP563xx_LoadVxD

```
BOOL DSP563xx_LoadVxD(pDSPINFO DSP563xx)
{
    if(!DSP563xx_CreateCommonEvent(&DSP563xx->Evnt3, &DSP563xx->Evnt0)) {
        return FALSE;
    }
    DSP563xx->DeviceHandle = CreateFile(DSP563xx->VxDFileName,
                                       0,0,0,CREATE_NEW, 0, 0);
    if (DSP563xx->DeviceHandle == INVALID_HANDLE_VALUE) {
        return FALSE;
    }
    return TRUE;
}
```

A.2 DSP563xx_UnLoadVxD

```
BOOL DSP563xx_UnLoadVxD(pDSPINFO DSP563xx)
{
    if (CloseHandle(DSP563xx->DeviceHandle)) {
        if (DeleteFile(DSP563xx->VxDFileName)) {
            return FALSE;
        }
        return TRUE;
    }
    return FALSE;
}
```

A.3 DSP563xx_InitializeDevice

```
BOOL DSP563xx_InitializeDevice(pDSPINFO DSP563xx)
{
    PVOID      InBuf[5];
    PVOID      OutBuf[3];
    DWORD      cbBytesReturned;
    DWORD      Status;

    InBuf[1] = DSP563xx->Evtnt0;
    InBuf[0] = (PVOID)INITIALIZATION_MESSAGE;
    InBuf[2] = (PVOID)DSP563xx->DeviceId;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    Status= (DWORD)OutBuf[0]; // Status returned by VxD

    if (Status == DEVNODE_NOT_FOUND) {
        return FALSE;
    }
    else {
        DSP563xx->HI32BaseAddress = (DWORD)OutBuf[1]; // base address for
hi32 mem space
        return TRUE;
    }
}
```

A.4 DSP563xx_ReadHI32Register

```
DWORD DSP563xx_ReadHI32Register(pDSPINFO DSP563xx , DWORD Register)
{
    DWORD* RegisterAddress;
    RegisterAddress = ((DWORD*)DSP563xx->HI32BaseAddress + Register);
    return *RegisterAddress;
}
```

A.5 DSP563xx_WriteHI32Register

```
void DSP563xx_WriteHI32Register(pDSPINFO DSP563xx , DWORD Register, DWORD
NewValue)
{
    DWORD* RegisterAddress;
    RegisterAddress = ((DWORD*)DSP563xx->HI32BaseAddress + Register);
    *RegisterAddress = NewValue;
}
```

A.6 DSP563xx_ReadCfgSpace

```
BOOL DSP563xx_ReadCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD* CfgWord)
{
    PVOID      InBuf[2];
    PVOID      OutBuf[2];
    DWORD      cbBytesReturned;
    DWORD*     Status;

    InBuf[0] = (PVOID)RD_CNFG_SPACE_MESSAGE;
    InBuf[1] = (PVOID)CfgOffset;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    Status = (DWORD*)OutBuf[0];
    if (Status) {
        *CfgWord = (DWORD)OutBuf[1];
        return TRUE;
    }
    else {
        return FALSE;
    }
}
```

A.7 DSP563xx_WriteCfgSpace

```
BOOL DSP563xx_WriteCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD CfgWord)
{
    PVOID      InBuf[3];
    PVOID      OutBuf[2];
    DWORD      cbBytesReturned;
    DWORD*     Status;

    InBuf[0] = (PVOID)WR_CNFG_SPACE_MESSAGE;
    InBuf[1] = (PVOID)CfgOffset;
    InBuf[2] = (PVOID)CfgWord;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    Status = (DWORD*)OutBuf[0];
    if (Status) {
        return TRUE;
    }
    else {
        return FALSE;
    }
}
```

A.8 DSP563xx_DownloadCode

```
BOOL DSP563xx_DownloadCode(pDSPINFO DSP563xx , const CHAR *Filename)
{
    BOOL ReturnValue;
    FILE *file;
    int i;
    DWORD NewValue;
    INT Size;
    DWORD Base;
    DWORD*DSPCode;
    DWORD Control;
    DWORD Checksum;
    DWORD Localsum;
    DSPCode = malloc(0x1000*sizeof(DWORD));

    ReturnValue = TRUE;
    Checksum= 0;
    Localsum= 0;
    // read code file (*.pci)
    file = fopen(Filename,"r");
    if(file == NULL )
    {
        ReturnValue = FALSE;
    }
    else
    {
        fscanf(file,"%8lx\n",&Size); // Base Address
        fscanf(file,"%8lx\n",&Base); // Number of dwords
        DSPCode[0] = Size;
        DSPCode[1] = Base;
        for (i=2;i<(Size+2);i++)
        {
            fscanf(file,"%8lx\n",&NewValue);
            DSPCode[i]= NewValue;
        }
        fclose(file);
    }
    // download code to DSP
    // resets HF[2:0]
    Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
    Control= (Control & ~(HCTR_HF0 | HCTR_HF1 | HCTR_HF2 ));
    DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
    // program data mode
    Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
    Control= Control & ~(HCTR_HRF1 | HCTR_HRF0 | HCTR_HTF1 | HCTR_HTF0);
    Control= Control | HCTR_HRF0 | HCTR_HTF0;
    DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
    // tx first and second words
    DSP563xx_WriteHI32Register(DSP563xx , HTXR_OFFSET, Size); // number of
words
    DSP563xx_WriteHI32Register(DSP563xx , HTXR_OFFSET, Base); // DSP memory
base address
```

```
    // tx code words
    for (i=2;i<(Size+2);i++)
    {
        DSP563xx_WriteHI32Register(DSP563xx , HTXR_OFFSET, DSPCode[i]); //
code word
        Localsum= Localsum + DSPCode[i];
    }
// END download code to DSP
Checksum= DSP563xx_ReadHI32Register(DSP563xx , HRXS_OFFSET);
Localsum = Localsum & 0x00ffffff;
if (Checksum!=Localsum)
{
    ReturnValue = FALSE;
}
else
{
    ReturnValue = TRUE;
}
free(DSPCode);

returnReturnValue;
}
```

A.9 DSP563xx_WaitForInterrupt

```
DWORD DSP563xx_WaitForInterrupt(pDSPINFO DSP563xx , DWORD Timeout)
{
    if (WaitForSingleObject((HANDLE)DSP563xx->Evt3, Timeout) ==
WAIT_TIMEOUT) {
        return WAIT_TIMEOUT;
    }
    return (0x00000038 & DSP563xx_ReadHI32Register(DSP563xx ,
HSTR_OFFSET)) >> 3 ;
}
```

A.10 DSP563xx_GetPhysAdd

```
DWORD DSP563xx_GetPhysAdd(DWORD LinAd, pDSPINFO DSP563xx)
{
    PVOID      InBuf[2];
    PVOID      OutBuf[2];
    DWORD      cbBytesReturned;
    DWORD      PhysicalAddress;
    DWORD      ReturnedLinearAddress;

    InBuf[0] = (PVOID)GET_PHYS_ADD_MESSAGE;
    InBuf[1] = (PVOID)LinAd;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    ReturnedLinearAddress = (DWORD)OutBuf[0]; // linear address
    PhysicalAddress = (DWORD)OutBuf[1]; // base address for a locked buffer

    return PhysicalAddress;
}
```

A.11 DSP563xx_LockMemoryPage

```
BOOL DSP563xx_LockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size)
{
    PVOID          InBuf[3];
    PVOID          OutBuf[2];
    DWORD          cbBytesReturned;
    DWORD          Status;

    InBuf[0] =(PVOID)LOCK_ONE_MEMORY_PAGE;
    InBuf[1] =(PVOID)LinearAddress;
    InBuf[2] =(PVOID)Size;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    return(DWORD)OutBuf[0];

    Status          =          (DWORD)OutBuf[0];

    printf ("LOCKED :: %8lx %8lx\n", Status, (DWORD)OutBuf[1]);

    if (Status) {
        return(DWORD)OutBuf[1];
    }
    else {
        returnFALSE;
    }
}
```

A.12 DSP563xx_UnLockMemoryPage

```
BOOL DSP563xx_UnLockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size)
{
    PVOID      InBuf[3];
    PVOID      OutBuf[2];
    DWORD      cbBytesReturned;

    InBuf[0] =(PVOID)UNLOCK_ONE_MEMORY_PAGE;
    InBuf[1] =(PVOID)LinearAddress;
    InBuf[2] =(PVOID)Size;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    return (BOOL)OutBuf[0]; // status returned by VxD
}
```

A.13 DSP563xx_ChangeDataMode

```
void DSP563xx_ChangeDataMode(pDSPINFO DSP563xx , DWORD HI32DataMode)
{
    DWORD Control;
    DWORD dwReturnValue;
    // resets HF[2:0]
    Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
    Control= (Control & ~(HCTR_HF0 | HCTR_HF1 | HCTR_HF2 ));
    DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);

    // send host command : Personal Reset
    DSP563xx_WriteHI32Register(DSP563xx , HCVR_OFFSET, 0xEF);
    // wait for acknowledge from the DSP
    dwReturnValue = DSP563xx_WaitForInterrupt((pDSPINFO)DSP563xx, INFINITE);
    Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
    switch(HI32DataMode) {
        case HI32_32BIT_MODE:
            // program data mode
            Control= Control & ~(HCTR_HRF1 | HCTR_HRF0 | HCTR_HTF1 |
HCTR_HTF0);
            DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
            break;
        case HI32_24BIT_MODE:
        default:
            Control= Control | HCTR_HRF0 | HCTR_HTF0;
            DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
    }
}
```

A.14 Auxiliary Functions

The two functions here are adapted from *Vireos VtoolsD* samples (© 1994-1995 Vireos Software,Inc):

```
// COPYRIGHT 1994,1995 Vireos Software, Inc.
HANDLE (WINAPI *(GetAddressOfOpenVxDHandle()))(HANDLE)
{
    CHAR K32Path[MAX_PATH];
    HINSTANCE hK32;

    GetSystemDirectory(K32Path, MAX_PATH);
    strcat(K32Path, "\\kernel32.dll");
    if ((hK32 = LoadLibrary(K32Path)) == 0)
        return NULL;

    return (HANDLE(WINAPI *) (HANDLE))GetProcAddress(hK32,
        "OpenVxDHandle");
}

// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions)
BOOL DSP563xx_CreateCommonEvent(HANDLE* Evnt3, HANDLE* Evnt0)
{
    static HANDLE (WINAPI *OpenVxD)(HANDLE)=0;

    *Evnt0 = 0;
    *Evnt3 = CreateEvent(0, FALSE, FALSE, NULL);

    if (OpenVxD == 0)
        OpenVxD = GetAddressOfOpenVxDHandle();

    if (OpenVxD && *Evnt3)
        *Evnt0 = OpenVxD(*Evnt3);
    else
        *Evnt0 = 0;

    return ( (*Evnt3 != 0) && (*Evnt0 != 0) );
}
```

A.15 DSP563xx_HI32_PCI Functions - C-Header file

```
//-----  
//  
//          COPYRIGHT 1998,1999 - MOTOROLA INC.  
//  
//-----  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <conio.h>  
#include <windows.h>  
#include <time.h>  
  
#define      HI32_USER_MESSAGE          1  
#define      DEVNODE_NOT_FOUND         0x00000010  
  
#define      INITIALIZATION_MESSAGE    1  
#define      LOCK_ONE_MEMORY_PAGE      2  
#define      RD_CNFG_SPACE_MESSAGE     3  
#define      WR_CNFG_SPACE_MESSAGE     4  
#define      GET_PHYS_ADD_MESSAGE      6  
#define      UNLOCK_ONE_MEMORY_PAGE    7  
  
#define HCTR_OFFSET      0x00000004  
#define HSTR_OFFSET      0x00000005  
#define HCVR_OFFSET      0x00000006  
#define HRXS_OFFSET      0x00000007  
#define HTXR_OFFSET      0x00000100  
#define HCTR_HF0         0x00000008  
#define HCTR_HF1         0x00000010  
#define HCTR_HF2         0x00000020  
#define HCTR_HTF0        0x00000100  
#define HCTR_HTF1        0x00000200  
#define HCTR_HRF0        0x00000800  
#define HCTR_HRF1        0x00001000  
#define HSTR_HRRQ        0x00000004  
  
#define HI32_24BIT_MODE  0x00400000  
#define HI32_32BIT_MODE  0x00000000  
  
typedef struct  
{  
    char* VxDFileName;  
    HANDLE DeviceHandle;  
    HANDLE Evnt0;  
    HANDLE Evnt3;  
    char* DSPName;  
    DWORD DeviceId;  
    DWORD VendorId;  
    DWORD HI32BaseAddress;  
  
} DSPINFO , *pDSPINFO;
```

```

// main functions
BOOL DSP563xx_LoadVxD(pDSPINFO DSP563xx);
BOOL DSP563xx_UnLoadVxD(pDSPINFO DSP563xx);
BOOL DSP563xx_InitializeDevice(pDSPINFO DSP563xx);
DWORD DSP563xx_ReadHI32Register(pDSPINFO DSP563xx , DWORDRegister);
void DSP563xx_WriteHI32Register(pDSPINFO DSP563xx , DWORDRegister, DWORD
NewValue);
BOOL DSP563xx_DownloadCode(pDSPINFO DSP563xx , const CHAR *Filename);
DWORD DSP563xx_WaitForInterrupt(pDSPINFO DSP563xx , DWORD Timeout);
BOOL DSP563xx_LockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size);
BOOL DSP563xx_UnLockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size);
void DSP563xx_ChangeDataMode(pDSPINFO DSP563xx , DWORD HI32DataMode);
BOOL DSP563xx_ReadCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD* Cfg-
Word);
BOOL DSP563xx_WriteCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD Cfg-
Word);
DWORD DSP563xx_GetPhysAdd(DWORD LinAd, pDSPINFO DSP563xx);
// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions of this function)
HANDLE (WINAPI *GetAddressOfOpenVxDHandle())(HANDLE);
// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions of this function)
BOOL DSP563xx_CreateCommonEvent(HANDLE* Evnt3, HANDLE* Evnt0);

```

```

//-----
//
//          COPYRIGHT 1998,1999 - MOTOROLA INC.
//
//-----

```

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
#include <time.h>

```

```

#define      HI32_USER_MESSAGE      1
#define      DEVNODE_NOT_FOUND      0x00000010

```

```

#define      INITIALIZATION_MESSAGE 1
#define      LOCK_ONE_MEMORY_PAGE   2
#define      RD_CNFG_SPACE_MESSAGE  3
#define      WR_CNFG_SPACE_MESSAGE  4
#define      GET_PHYS_ADD_MESSAGE    6
#define      UNLOCK_ONE_MEMORY_PAGE  7

```

```

#define HCTR_OFFSET      0x00000004
#define HSTR_OFFSET      0x00000005
#define HCVR_OFFSET      0x00000006
#define HRXS_OFFSET      0x00000007
#define HTXR_OFFSET      0x00000100
#define HCTR_HF0         0x00000008

```

```
#define HCTR_HF1          0x00000010
#define HCTR_HF2          0x00000020
#define HCTR_HTF0        0x00000100
#define HCTR_HTF1        0x00000200
#define HCTR_HRF0        0x00000800
#define HCTR_HRF1        0x00001000
#define HSTR_HRRQ        0x00000004
```

```
#define HI32_24BIT_MODE 0x00400000
#define HI32_32BIT_MODE 0x00000000
```

```
typedef struct
```

```
{
    char* VxDFileName;
    HANDLE DeviceHandle;
    HANDLE Evnt0;
    HANDLE Evnt3;
    char* DSPName;
    DWORD DeviceId;
    DWORD VendorId;
    DWORD HI32BaseAddress;

```

```
} DSPINFO , *pDSPINFO;
```

```
// main functions
```

```
BOOL DSP563xx_LoadVxD(pDSPINFO DSP563xx);
BOOL DSP563xx_UnLoadVxD(pDSPINFO DSP563xx);
BOOL DSP563xx_InitializeDevice(pDSPINFO DSP563xx);
DWORD DSP563xx_ReadHI32Register(pDSPINFO DSP563xx , DWORD Register);
void DSP563xx_WriteHI32Register(pDSPINFO DSP563xx , DWORD Register, DWORD
NewValue);
BOOL DSP563xx_DownloadCode(pDSPINFO DSP563xx , const CHAR *Filename);
DWORD DSP563xx_WaitForInterrupt(pDSPINFO DSP563xx , DWORD Timeout);
BOOL DSP563xx_LockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size);
BOOL DSP563xx_UnLockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size);
void DSP563xx_ChangeDataMode(pDSPINFO DSP563xx , DWORD HI32DataMode);
BOOL DSP563xx_ReadCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD* Cfg-
Word);
BOOL DSP563xx_WriteCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD Cfg-
Word);
DWORD DSP563xx_GetPhysAdd(DWORD LinAd, pDSPINFO DSP563xx);
// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions of this function)
HANDLE (WINAPI *GetAddressOfOpenVxDHandle())(HANDLE);
// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions of this function)
BOOL DSP563xx_CreateCommonEvent(HANDLE* Evnt3, HANDLE* Evnt0);
```

B Code Listing

This appendix presents the source C-code for the Virtual Device Driver (VxD), as well as the DSP56301 assembly code for the sample routines that support the DSP563xx_HI32_PCI functions stimuli. The listing of the source C-code for the DSP56301 example of usage of DSP563xx_HI32_PCI is also provided.

B.1 Virtual Device Code - VxD Source - C-code

```
// DSPVXD.c
//-----
//
//          COPYRIGHT 1998,1999 - MOTOROLA INC.
//-----

#define    DEVICE_MAIN
#include  "dspvxd.h"
#undef    DEVICE_MAIN

Declare_Virtual_Device(DSPVXD)

// Handle of Synchronization Event between App/VxD
HANDLE    CommonEvent;
// points to ADS56301/HI32 device node in Win95 Reg
DEVNODE    HI32DeviceNode;
CONFIGRETRetValue;
// Buffer for HI32's Logical Configuration
CMCONFIGHI32LogicalConfiguration;
// Physical Page Add of Base Add of HI32 Memory Space
DWORD    HI32MemSpaceFirstPage;
// Linear Page Add of Base Add of HI32 Memory Space
DWORD    HI32MemSpaceLinAddr;
// Handle for virtual IRQ
IRQHANDLEHI32_IRQHandle;
// Thunk for interrupt handler
VPICD_HWInt_THUNK HI32_Int_Thunk;
// Thunk for event handler
PriorityVMEEvent_THUNK HI32_EventThunk;

DWORD    Message;
CHAR    ID2[23];
DWORD    HI32MemSpaceLinAdLocked;
DWORD    LockedLinearAddress;
DWORD*    HCVRAAddress;
DWORD*    HSTRAddress;

////////////////////////////////////
//
VOID SearchHWTTree(DEVNODE Node, DEVNODE* TargetNode)
{
```



```

        if (CommonEvent) {
            _VWIN32_SetWin32Event(CommonEvent);
        }
    }

    ////////////////////////////////////////////////////////////////////
    // Interrupt Handler
    BOOL __stdcall HI32_Int_Handler(VMHANDLE hVM, IRQHANDLE hIRQ)
    {
        // check if HINTA bit is set and service the interrupt if it is from the
        DSP
        HSTRAddress= (DWORD*)(HI32MemSpaceLinAdLocked) + 0x5;
        if (*HSTRAddress & 0x00000040) {
            // send Host Command for deasserting INTA
            HCVRAAddress= (DWORD*)(HI32MemSpaceLinAdLocked) + 0x6;
            *HCVRAAddress = 0x000080ff;
            // wait until HINTA bit is reset
            HSTRAddress= (DWORD*)(HI32MemSpaceLinAdLocked) + 0x5;
            while (HSTR & 0x00000040) {
                HSTR = *HSTRAddress;
            }
            // tell VPICD to clear the interrupt
            VPICD_Phys_EOI(HI32_IRQHandle);
            // signal to App that the interrupt has been received and acknowl-
            edged
            Call_Priority_VM_Event(TIME_CRITICAL_BOOST,
                                   Get_Sys_VM_Handle(),
                                   0,
                                   NULL,
                                   EventService,
                                   0,
                                   &HI32_EventThunk
                                   );
        }
        return TRUE;
    }
    ////////////////////////////////////////////////////////////////////
    BOOL InterruptEnable()
    {
        // struct to pass to VPICD_Virtualize_IRQ
        struct VPICD_IRQ_Descriptor IRQdesc;
        // Fill up the structure to pass to VPICD_Virtualize_IRQ
        // IRQ to virtualize
        IRQdesc.VID_IRQ_Number = (DWORD)HI32LogicalConfiguration.bIRQRegis-
        ters[0];
        // Flags
        IRQdesc.VID_Options = 0x17;
        // set address of handler
        IRQdesc.VID_Hw_Int_Proc =
            (DWORD)VPICD_Thunk_HWInt(HI32_Int_Handler, &HI32_Int_Thunk);
        // The other callbacks are not used.
        IRQdesc.VID_Virt_Int_Proc = 0;
        IRQdesc.VID_EOI_Proc = 0;
        IRQdesc.VID_Mask_Change_Proc = 0;
    }

```

```

    IRQdesc.VID_IRET_Proc = 0;
    // Now pass the structure to VPICD. VPICD returns the IRQ handle.
    HI32_IRQHandle = VPICD_Virtualize_IRQ(&IRQdesc);
    // unmask IRQ
    VPICD_Physically_Unmask(HI32_IRQHandle);
    return TRUE;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
DefineControlHandler(SYS_DYNAMIC_DEVICE_INIT, OnSysDynamicDeviceInit);
DefineControlHandler(SYS_DYNAMIC_DEVICE_EXIT, OnSysDynamicDeviceExit);
DefineControlHandler(W32_DEVICEIOCONTROL, OnW32Deviceiocontrol);

BOOL __cdecl ControlDispatcher(
    DWORD dwControlMessage,
    DWORD EBX,
    DWORD EDX,
    DWORD ESI,
    DWORD EDI,
    DWORD ECX)
{
    START_CONTROL_DISPATCH

        ON_SYS_DYNAMIC_DEVICE_INIT(OnSysDynamicDeviceInit);
        ON_SYS_DYNAMIC_DEVICE_EXIT(OnSysDynamicDeviceExit);
        ON_W32_DEVICEIOCONTROL(OnW32Deviceiocontrol);

    END_CONTROL_DISPATCH

    return TRUE;
}

BOOL OnSysDynamicDeviceInit()
{
    CommonEvent = 0;
    return TRUE;
}

BOOL OnSysDynamicDeviceExit()
{
    if (CommonEvent){
        _VWIN32_CloseVxDHandle(CommonEvent);
    }
    VPICD_Physically_Mask(HI32_IRQHandle);
    VPICD_Force_Default_Behavior(HI32_IRQHandle);
    LinPageUnLock(HI32MemSpaceLinAdLocked >> 12, 1, PAGEMAPGLOBAL);

    return TRUE;
}

DWORD OnW32Deviceiocontrol(PIOCTLPARAMS p)
{
    DWORD          CfgOffset;
    DWORD*         CfgBuf;

```

```

DWORD      CfgBuf2;
DWORD      DevID;
DWORD      VenID;
CHAR       cDevID[5];
CHAR       cVenID[5];
DWORD      LinearAddress;
DWORD      Size;
DWORD      PhysicalAddress;
DWORD      Status;

switch (p->dioc_IOCTLCode)
{
    case DIOC_OPEN:
    case DIOC_CLOSEHANDLE:

        return 0;

    case HI32_USER_MESSAGE:

        Message      =      ((DWORD*)(p->dioc_InBuf))[0];
        if (Message) {
            switch (Message) {
                case INITIALIZATION_MESSAGE:// Initialization
                    Message

                    CommonEvent = ((HANDLE*)p->dioc_InBuf)[1];
                    LinPageUnLock(HI32MemSpaceLinAdLocked, 1,
                    PAGEMAPGLOBAL);

                    VenID          = 0x1057;
                    DevID          =

                    ((DWORD*)(p->dioc_InBuf))[2];

                    strcpy (ID2, "PCI\\VEN_");
                    _ultoa(VenID, cVenID, 16);
                    strcat(ID2, cVenID);
                    strcat (ID2, "&DEV_");
                    _ultoa(DevID, cDevID, 16);
                    strcat(ID2, cDevID);
                    Status = 0;
                    if ( Initial() ) {
                        InterruptEnable();
                        ((DWORD*)(p->dioc_OutBuf))[1] =
                    HI32MemSpaceLinAdLocked;

                    }
                    else {
                        Status = (Status |
                    DEVNODE_NOT_FOUND);

                        ((DWORD*)(p->dioc_OutBuf))[1] =
                    (DWORD)0;

                    }
                    ((DWORD*)(p->dioc_OutBuf))[0] = Status;
                    *p->dioc_bytesret = 2*sizeof(DWORD);

                    return 0;

```

```

        case LOCK_ONE_MEMORY_PAGE:// Lock Buffer Pages
            LinearAddress=
((DWORD*)(p->dioc_InBuf))[1];
            Size
            =
((DWORD*)(p->dioc_InBuf))[2];
            Size = ((LinearAddress + Size - 1) >> 12)
- (LinearAddress >> 12) + 1;
            LockedLinearAddress=
            Size=
            IsPageLock(LinearAddress
>> 12,Size, 0);
            if (LockedLinearAddress) {
                ((DWORD*)(p->dioc_OutBuf))[0] =
                ((DWORD*)Size;
            }
            else {
                ((DWORD*)(p->dioc_OutBuf))[0] =
                ((DWORD*)0;
            }
            ((DWORD*)(p->dioc_OutBuf))[1] = LockedLin-
earAddress;
            *p->dioc_bytesret = 2*sizeof(DWORD);
            return 0;

        case RD_CNFG_SPACE_MESSAGE:// Read ConfigSpace
            CfgOffset =
((DWORD*)(p->dioc_InBuf))[1];
            CfgBuf = 0;
            ReturnValue =
CONFIGMG_Call_Enumerator_Function(HI32DeviceNode,0,CfgOffset,&CfgBuf,0x4,0);
            if (ReturnValue == CR_SUCCESS) {
                ((BOOL*)(p->dioc_OutBuf))[0] =
                ((BOOL)TRUE;
            }
            else {
                ((BOOL*)(p->dioc_OutBuf))[0] =
                ((BOOL)FALSE;
            }
            ((DWORD*)(p->dioc_OutBuf))[1] =
            ((DWORD)CfgBuf;
            *p->dioc_bytesret = 2*sizeof(DWORD);
            return 0;

        case WR_CNFG_SPACE_MESSAGE:// Write ConfigSpace
            CfgOffset =
((DWORD*)(p->dioc_InBuf))[1];
            CfgBuf2 =
            ((DWORD*)(p->dioc_InBuf))[2];
            ReturnValue =
CONFIGMG_Call_Enumerator_Function(HI32DeviceNode,1,CfgOffset,&CfgBuf2,0x4,0);
            if (ReturnValue == CR_SUCCESS) {
                ((BOOL*)(p->dioc_OutBuf))[0] =
                ((BOOL)TRUE;
            }
            else {
                ((BOOL*)(p->dioc_OutBuf))[0] =
                ((BOOL)FALSE;
            }

```

```

else {
    ((BOOL*)(p->dioc_OutBuf))[0] =
(BOOL)FALSE;
}
((DWORD*)(p->dioc_OutBuf))[1] =
(DWORD)*CfgBuf;
*p->dioc_bytesret = 2*sizeof(DWORD);
return 0;

case GET_PHYS_ADD_MESSAGE:// Convert Linear
Address to Physical Address
// It is supposed that the correspondent
page was previously locked (Message 2)
// Get Linear Address from App
LinearAddress=
((DWORD*)(p->dioc_InBuf))[1];
// Retrieve correspondent Physical Address
CopyPageTable(LinearAddress>>
12,1,(PPVOID)&PhysicalAddress,0);
PhysicalAddress=((PhysicalAddress &
0xfffff000) | (LinearAddress & 0x0fff));
((DWORD*)(p->dioc_OutBuf))[0] = LinearAd-
dress;
((DWORD*)(p->dioc_OutBuf))[1] = Physical-
Address;
*p->dioc_bytesret = 2*sizeof(DWORD);
return 0;

case UNLOCK_ONE_MEMORY_PAGE:// Unlock Buffer
Pages (those which were locked in case 2)
LinearAddress=
((DWORD*)(p->dioc_InBuf))[1]; // Linear Address from App
Size =
((DWORD*)(p->dioc_InBuf))[2]; // Size of Buffer, in bytes
Size = ((LinearAddress + Size - 1) >> 12)
- (LinearAddress >> 12) + 1;
((BOOL*)(p->dioc_OutBuf))[0] = LinPageUn-
Lock(LinearAddress >> 12, Size, 0);
*p->dioc_bytesret = sizeof(BOOL);
return 0;

} // switch (Message)

} // if (CommonEvent)
return 0;

default:
return -1;
} // switch (p->dioc_IOCTLCode)
return 0;
}

```

B.2 Virtual Device Code - VxD Source - C-Header file

```
// DSPVXD.h - include file for VxD DSPVXD
//-----
//
//          COPYRIGHT 1998,1999 - MOTOROLA INC.
//
//-----

#include <vtoolsc.h>

#define     DSPVXD_Major      1
#define     DSPVXD_Minor     0
#define     DSPVXD_DeviceID  UNDEFINED_DEVICE_ID
#define     DSPVXD_Init_Order UNDEFINED_INIT_ORDER

#define     HI32_USER_MESSAGE 1

#define     DEVNODE_NOT_FOUND 0x00000010

#define     INITIALIZATION_MESSAGE 1
#define     LOCK_ONE_MEMORY_PAGE   2
#define     RD_CNFG_SPACE_MESSAGE  3
#define     WR_CNFG_SPACE_MESSAGE  4
#define     GET_PHYS_ADD_MESSAGE   6
#define     UNLOCK_ONE_MEMORY_PAGE 7
```

B.3 DSP56301 Assembly Code

```
;-----
;
;          COPYRIGHT 1998 , 1999 - MOTOROLA INC.
;
;-----

    page    132,55,0,0,0
    nolist
    INCLUDE "/home/sabres/nde/docs/sio/ioequ.asm"
    INCLUDE "/home/sabres/nde/docs/sio/intequ.asm"
    list

;-----
; EQUATES
;-----

START          equ $100
HOST_COMMAND_EF equ $ee      ; Host Command Interrupt Vector Address
HOST_COMMAND_F1 equ $f0      ; Host Command Interrupt Vector Address
HOST_COMMAND_F3 equ $f2      ; Host Command Interrupt Vector Address
HOST_COMMAND_F5 equ $f4      ; Host Command Interrupt Vector Address
HOST_COMMAND_F7 equ $f6      ; Host Command Interrupt Vector Address
HOST_COMMAND_F9 equ $f8      ; Host Command Interrupt Vector Address
HOST_COMMAND_FB equ $fa      ; Host Command Interrupt Vector Address
HOST_COMMAND_FD equ $fc      ; Host Command Interrupt Vector Address
```

```

HOST_COMMAND_FF equ $fe          ; Host Command Interrupt Vector Address
;-----
; start of program area
;-----
    org p:I_RESET                ;Hardware RESET
    jmp >START

    dup (I_INTEND-#+1)           ;fill vector space
    jmp <+
endm

; Interrupt Vectors

; Host Commands Interrupts
    org P:HOST_COMMAND_EF        ; Personal Reset: Mode 0 -> HACT = 0 -> PCI Mode
    jsr <personal_reset
    nop

    org P:HOST_COMMAND_F1; set interrupt index to $2
    bset #3,x:M_DCTR
    nop
    org P:HOST_COMMAND_F3; set interrupt index to $3
    bclr #3,x:M_DCTR
    nop
    org P:HOST_COMMAND_F5; set interrupt index to $4
    bset #4,x:M_DCTR
    nop
    org P:HOST_COMMAND_F7; set interrupt index to $5
    bclr #4,x:M_DCTR
    nop
    org P:HOST_COMMAND_F9; set interrupt index to $6
    bset #5,x:M_DCTR
    nop
    org P:HOST_COMMAND_FB; set interrupt index to $7
    bclr #5,x:M_DCTR
    nop

    org P:HOST_COMMAND_FD
    bset #6,x:M_DCTR ; ASSERT HI32 PCI interrupt line (HINTA)
    nop

    org P:HOST_COMMAND_FF
    bclr #6,x:M_DCTR ; Deassert HI32 PCI interrupt line (HINTA)
    nop

    org p:(I_INTEND+1)
    dup (START-I_INTEND-1) ;fill with nops
    nop
endm
;-----
    org P:START

    move #0,x1                ;clear N1 for flagging
    move #0,sr                 ; enable interrupts

```

```

        movep    #$000003,x:M_IPRP    ; HI32's IPL=2
        movep    #$03f000,x:M_IPRC    ; DMA's IPL=2, channels #0 and #1 and #2

;Program DMA channel #0 for output
        movep    #$500,x:M_DSR0
        movep    #>M_DTXS,x:M_DDR0
        movep    #(-$1f),x:M_DOR0
        movep    #$00001f,x:M_DCO0

;Program DMA channel #1 for input
        movep    #>M_DRXR,x:M_DSR1
        movep    #$700,x:M_DDR1
        movep    #(-$1f),x:M_DOR1
        movep    #$00001f,x:M_DCO1

; PCI personal reset, HI32 PCI-mode, HCIE set, MACE=1 , MAIE = 1

        movep    #>$000000,x:M_DCTR ; HM=$0 (Personal s/w reset)
        nop
        nop
        jset    #M_HACT,x:M_DSR,*      ; wait for personal reset
        movep    #>$000000,x:M_DPCR ;
        movep    #>$100001,x:M_DCTR ; HM=$1 ,HCIE=$1

; sum up checksum and sends to HOST

        move     #$0,r1
        clr     a
        clr     b
        do     #(the_end+1),loop1
        move    p:(r1)+,b1
        add    b,a
        nop
        nop
loop1
        nop
        nop
wait_for_request
        brclr   #M_STRQ,x:M_DSR,wait_for_request ; Write data to FIFO
        movep   a1,x:M_DTXS
        nop

;Activate DMA Channel #0 for OUTPUT
;          DDDDDDDDDDDD DDDDDDDDDDD
;          DTTTPPCRRRRRDAAAAAADSS
;          DIMMMRROSSSSS3MMMMMMSSSS
;          EE21010N43210D5432101010
;          |||||||||||||||||||||||
        movep   %#101011101111001000000010,x:M_DCR0
;Activate DMA Channel #1 for INPUT
;          DDDDDDDDDDDD DDDDDDDDDDD
;          DTTTPPCRRRRRDAAAAAADSS
;          DIMMMRROSSSSS3MMMMMMSSSS
;          EE21010N43210D5432101010

```

```

;          |||
movep    #%101011101110000011001000,x:M_DCR1
nop
;-----
personal_reset
; PCI personal reset, HI32 PCI-mode, HCIE set, MACE=1 , MAIE = 1
movep    #>$000000,x:M_DCTR ; HM=$0 (Personal s/w reset)
nop
nop
jset     #M_HACT,x:M_DSR,* ; wait for personal reset
movep    #>$100001,x:M_DCTR ; HM=$1 ,HCIE=$1
; send interrupt #1
brset    #M_HINT,x:M_DCTR,* ; polling for no int is pending
bset     #M_HF3,x:M_DCTR
bclr     #M_HF4,x:M_DCTR
bclr     #M_HF5,x:M_DCTR
bset     #M_HINT,x:M_DCTR ; assert HINTA
; return from interrupt
nop
rti
;-----
org P:$500 ; output data
dc       $fe0000
dc       $fe0001
dc       $fe0002
dc       $fe0003
dc       $fe0004
dc       $fe0005
dc       $fe0006
dc       $fe0007
dc       $fe0008
dc       $fe0009
dc       $fe000a
dc       $fe000b
dc       $fe000c
dc       $fe000d
dc       $fe000e
dc       $fe000f
dc       $fe0010
dc       $fe0011
dc       $fe0012
dc       $fe0013
dc       $fe0014
dc       $fe0015
dc       $fe0016
dc       $fe0017
dc       $fe0018

```

```

dc    $fe0019
dc    $fe001a
dc    $fe001b
dc    $fe001c
dc    $fe001d
dc    $fe001e
dc    $fe001f

org P:$600    ; input data
nop

org P:$800
nop
the_end

```

B.4 DSP56301 Sample - C-code

```

//-----
//
//          COPYRIGHT 1998,1999 - MOTOROLA INC.
//
//-----
#include "dsp56301.h"

////////////////////////////////////////////////////////////////
void DSP563xx_ChangeDataMode(pDSPINFO DSP563xx , DWORD HI32DataMode)
{
    DWORD Control;
    DWORD dwReturnValue;
    // resets HF[2:0]
    Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
    Control= (Control & ~(HCTR_HF0 | HCTR_HF1 | HCTR_HF2 ));
    DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);

    // send host command : Personal Reset
    DSP563xx_WriteHI32Register(DSP563xx , HCVR_OFFSET, 0xEF);
    // wait for acknowledge from the DSP
    dwReturnValue = DSP563xx_WaitForInterrupt((pDSPINFO)DSP563xx, INFINITE);
    Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
    switch(HI32DataMode) {
        case HI32_32BIT_MODE:
            // program data mode
            Control= Control & ~(HCTR_HRF1 | HCTR_HRF0 | HCTR_HTF1 |
HCTR_HTF0);
            DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
            break;
        case HI32_24BIT_MODE:
        default:
            Control= Control | HCTR_HRF0 | HCTR_HTF0;
            DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
    }
}
////////////////////////////////////////////////////////////////

```

```

////////////////////////////////////
BOOL DSP563xx_LoadVxD(pDSPINFO DSP563xx)
{
    if(!DSP563xx_CreateCommonEvent(&DSP563xx->Evt3, &DSP563xx->Evt0)) {
        return FALSE;
    }
    DSP563xx->DeviceHandle = CreateFile(DSP563xx->VxDFileName,
0,0,0,CREATE_NEW, 0, 0);
    if (DSP563xx->DeviceHandle == INVALID_HANDLE_VALUE)
    {
        return FALSE;
    }
    return TRUE;
}
////////////////////////////////////
////////////////////////////////////
BOOL DSP563xx_UnLoadVxD(pDSPINFO DSP563xx)
{
    if (CloseHandle(DSP563xx->DeviceHandle))
    {
        if (DeleteFile(DSP563xx->VxDFileName))
        {
            return FALSE;
        }
        return TRUE;
    }
    return FALSE;
}
////////////////////////////////////
////////////////////////////////////
BOOL DSP563xx_InitializeDevice(pDSPINFO DSP563xx)
{
    PVOID      InBuf[5];
    PVOID      OutBuf[3];
    DWORD      cbBytesReturned;
    DWORD      Status;

    InBuf[1] =DSP563xx->Evt0;
    InBuf[0] =(PVOID)INITIALIZATION_MESSAGE;
    InBuf[2] =(PVOID)DSP563xx->DeviceId;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE,InBuf,
        sizeof(PVOID),(LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    Status= (DWORD)OutBuf[0];// Status returned by VxD

    if (Status == DEVNODE_NOT_FOUND) {
        returnFALSE;
    }
    else {
        DSP563xx->HI32BaseAddress = (DWORD)OutBuf[1];// base address for
hi32 mem space

```

```

        return TRUE;
    }
}
//
//
void DSP563xx_WriteHI32Register(pDSPINFO DSP563xx , DWORD Register, DWORD
NewValue)
{
    DWORD* RegisterAddress;
    RegisterAddress = ((DWORD*)DSP563xx->HI32BaseAddress + Register);
    *RegisterAddress = NewValue;
}
//
//
DWORD DSP563xx_ReadHI32Register(pDSPINFO DSP563xx , DWORD Register)
{
    DWORD* RegisterAddress;
    RegisterAddress = ((DWORD*)DSP563xx->HI32BaseAddress + Register);
    return *RegisterAddress;
}

//
//
BOOL DSP563xx_DownloadCode(pDSPINFO DSP563xx , const CHAR *Filename)
{
    BOOL ReturnValue;
    FILE *file;
    int i;
    DWORD NewValue;
    INT Size;
    DWORD Base;
    DWORD* DSPCode;
    DWORD Control;
    DWORD Checksum;
    DWORD Localsum;
    DSPCode = malloc(0x1000*sizeof(DWORD));

    ReturnValue = TRUE;
    Checksum= 0;
    Localsum= 0;
    // read code file (*.pci)
    file = fopen(Filename,"r");
    if(file == NULL )
    {
        ReturnValue = FALSE;
    }
    else
    {
        fscanf(file,"%8lx\n",&Size); // Base Address
        fscanf(file,"%8lx\n",&Base); // Number of dwords
        DSPCode[0] = Size;
        DSPCode[1] = Base;
        for (i=2;i<(Size+2);i++)
        {

```

```

        fscanf(file,"%8lx\n",&NewValue);
        DSPCode[i]= NewValue;
    }
    fclose(file);
}
// download code to DSP
// resets HF[2:0]
Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
Control= (Control & ~(HCTR_HF0 | HCTR_HF1 | HCTR_HF2 ));
DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
// program data mode
Control= DSP563xx_ReadHI32Register(DSP563xx , HCTR_OFFSET);
Control= Control & ~(HCTR_HRF1 | HCTR_HRF0 | HCTR_HTF1 | HCTR_HTF0);
Control= Control | HCTR_HRF0 | HCTR_HTF0;
DSP563xx_WriteHI32Register(DSP563xx , HCTR_OFFSET, Control);
// tx first and second words
DSP563xx_WriteHI32Register(DSP563xx , HTXR_OFFSET, Size); // number of
words
    DSP563xx_WriteHI32Register(DSP563xx , HTXR_OFFSET, Base); // DSP memory
base address
    // tx code words
    for (i=2;i<(Size+2);i++)
    {
        DSP563xx_WriteHI32Register(DSP563xx , HTXR_OFFSET, DSPCode[i]); //
code word
        Localsum= Localsum + DSPCode[i];
    }
// END download code to DSP
Checksum= DSP563xx_ReadHI32Register(DSP563xx , HRXS_OFFSET);
Localsum = Localsum & 0x00ffffff;
if (Checksum!=Localsum)
{
    ReturnValue = FALSE;
}
else
{
    ReturnValue = TRUE;
}
free(DSPCode);

returnReturnValue;
}

////////////////////////////////////
////////////////////////////////////
DWORD DSP563xx_WaitForInterrupt(pDSPINFO DSP563xx , DWORD Timeout)
{
    if (WaitForSingleObject((HANDLE)DSP563xx->Evnt3, Timeout) ==
WAIT_TIMEOUT) {
        return WAIT_TIMEOUT;
    }
    return (0x00000038 & DSP563xx_ReadHI32Register(DSP563xx , HSTR_OFFSET))
>> 3 ;
}

```

```

{
    PVOID          InBuf[2];
    PVOID          OutBuf[2];
    DWORD          cbBytesReturned;
    DWORD*         Status;

    InBuf[0] =(PVOID)RD_CNFG_SPACE_MESSAGE;
    InBuf[1] =(PVOID)CfgOffset;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    Status= (DWORD*)OutBuf[0];
    if (Status) {
        *CfgWord= (DWORD)OutBuf[1];
        returnTRUE;
    }
    else {
        returnFALSE;
    }
}
//
//
BOOL DSP563xx_WriteCfgSpace(DWORD CfgOffset, pDSPINFO DSP563xx, DWORD CfgWord)
{
    PVOID          InBuf[3];
    PVOID          OutBuf[2];
    DWORD          cbBytesReturned;
    DWORD*         Status;

    InBuf[0] =(PVOID)WR_CNFG_SPACE_MESSAGE;
    InBuf[1] =(PVOID)CfgOffset;
    InBuf[2] =(PVOID)CfgWord;

    DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
        sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
        &cbBytesReturned, NULL);

    Status= (DWORD*)OutBuf[0];
    if (Status) {
        CfgWord= (DWORD)OutBuf[1];
        returnTRUE;
    }
    else {
        returnFALSE;
    }
}
//
//
DWORD DSP563xx_GetPhysAdd(DWORD LinAd, pDSPINFO DSP563xx)
{
    PVOID          InBuf[2];
    PVOID          OutBuf[2];

```

```

DWORD      cbBytesReturned;
DWORD      PhysicalAddress;
DWORD      ReturnedLinearAddress;

InBuf[0] = (PVOID)GET_PHYS_ADD_MESSAGE;
InBuf[1] = (PVOID)LinAd;

DeviceIoControl(DSP563xx->DeviceHandle, HI32_USER_MESSAGE, InBuf,
                sizeof(PVOID), (LPVOID)OutBuf, sizeof(OutBuf),
                &cbBytesReturned, NULL);

ReturnedLinearAddress = (DWORD)OutBuf[0]; // linear address
PhysicalAddress = (DWORD)OutBuf[1]; // base address for a locked buffer

returnPhysicalAddress;
}

////////////////////////////////////
////////////////////////////////////
// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions of this function)
HANDLE (WINAPI *(GetAddressOfOpenVxDHandle()))(HANDLE)
{
    CHAR K32Path[MAX_PATH];
    HINSTANCE hK32;

    GetSystemDirectory(K32Path, MAX_PATH);
    strcat(K32Path, "\\kernel32.dll");
    if ((hK32 = LoadLibrary(K32Path)) == 0)
        return NULL;

    return (HANDLE(WINAPI *) (HANDLE))GetProcAddress(hK32,
        "OpenVxDHandle");
}

////////////////////////////////////
////////////////////////////////////
// COPYRIGHT 1994,1995 Vireos Software, Inc. (this function)
BOOL DSP563xx_CreateCommonEvent(HANDLE* Evnt3, HANDLE* Evnt0)
{
    static HANDLE (WINAPI *OpenVxD)(HANDLE)=0;

    *Evnt0 = 0;
    *Evnt3 = CreateEvent(0, FALSE, FALSE, NULL);

    if (OpenVxD == 0)
        OpenVxD = GetAddressOfOpenVxDHandle();

    if (OpenVxD && *Evnt3)
        *Evnt0 = OpenVxD(*Evnt3);
    else
        *Evnt0 = 0;

    return ( (*Evnt3 != 0) && (*Evnt0 != 0) );
}

```

```

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
main()
{
////////////////////////////////////
// Some auxiliary variables
////////////////////////////////////
    DWORD*LinearAddress;

    BOOL  bReturnValue;
    DWORD dwReturnValue;
    DWORD dwNewValue;
////////////////////////////////////
// Chip Declaration
////////////////////////////////////
    pDSPINFO DSP56301;
    DSP56301 = malloc(0x400);
    DSP56301->DeviceId = 0x1801;
    DSP56301->VxDFileName = "\\.\DSPVXD.VXD";
////////////////////////////////////
    printf("*****.\n\n");
    printf("\t\tMOTOROLA INC.\n\n");
    printf("\t\tCOPYRIGHT 1999\n\n");
    printf("\tSample Usage of DSP563xx_PCI_FUNCTIONS\n\n");
    printf("*****.\n\n");
////////////////////////////////////
// Example of DSP563xx_LoadVxD
////////////////////////////////////
    printf("\nLoading %s : ",DSP56301->VxDFileName);
    bReturnValue = DSP563xx_LoadVxD(DSP56301);
    if (bReturnValue)
    {
        printf(" Passed\n");
    }
    else
    {
        printf(" Failed\n");
    }
////////////////////////////////////
// Example of DSP563xx_InitializeDevice
////////////////////////////////////
    bReturnValue = DSP563xx_InitializeDevice(DSP56301);
    if (bReturnValue)
    {
        printf("\nDevice Initialized. HI32 Memory Base Address:
%08lx\n",DSP56301->HI32BaseAddress);
    }
    else
    {
        printf(" Failed Initialize Device\n");
        exit(1);
    }
}

```

```

////////////////////////////////////
// Example of DSP563xx_DownloadCode
////////////////////////////////////
printf ("\nDownloading code to DSP: ");
bReturnValue = DSP563xx_DownloadCode(DSP56301, "dsp56301.pci");
if (bReturnValue) {
    printf(" Passed\n");
}
else {
    printf(" Failed\n");
}
////////////////////////////////////
// Example of DSP563xx_ReadHI32Register
////////////////////////////////////
printf ("\nReading Registers...\n\n");
printf ("\tHCTR : %08lx\t", DSP563xx_ReadHI32Register(DSP56301 ,
HCTR_OFFSET));
printf ("\tHSTR : %08lx\t", DSP563xx_ReadHI32Register(DSP56301 ,
HSTR_OFFSET));
printf ("\tHCVR : %08lx\n", DSP563xx_ReadHI32Register(DSP56301 ,
HCVR_OFFSET));
////////////////////////////////////
// Example of DSP563xx_WriteHI32Register
////////////////////////////////////
printf ("\nWriting to HCTR : Setting Host Flags.\n\n");
dwReturnValue = DSP563xx_ReadHI32Register(DSP56301 , HCTR_OFFSET);
printf ("\tOld Value of HCTR: %08lx\t", dwReturnValue);
dwReturnValue = dwReturnValue | 0x00000038; // set Host Flags HF2..HF0
DSP563xx_WriteHI32Register(DSP56301 , HCTR_OFFSET, dwReturnValue);
printf ("\tNew Value of HCTR:
%08lx\n", DSP563xx_ReadHI32Register(DSP56301 , HCTR_OFFSET));
printf ("\nPress <Space> to continue.");
while (!kbhit()){};
    _getch();

printf("\n*****\n");
////////////////////////////////////
// Example of Read/Write Config Space
////////////////////////////////////
printf("\nReading and Writing Configuration Space: clearing BM
bit\n\n");
bReturnValue = DSP563xx_ReadCfgSpace(0x4, DSP56301, &dwReturnValue );
if (bReturnValue) {
    printf("\tOld Value CSTR/CCMR: %08lx\t", dwReturnValue);
    dwNewValue = dwReturnValue & 0xfffffff; // clear BM bit
    bReturnValue = DSP563xx_WriteCfgSpace(0x04, DSP56301, dwNewValue
);
    if (bReturnValue) {
        bReturnValue = DSP563xx_ReadCfgSpace(0x4, DSP56301, &dwRe-
turnValue );
        if (bReturnValue){
            printf("\tNew Value CSTR/CCMR: %08lx\n", dwReturnValue);
        }
    }
}

```

```

        else {
            printf("\tFailed to Read Configuration Space\n");
        }
    }
    else {
        printf("\tFailed to Write Configuration Space\n");
    }
}
else {
    printf("\tFailed to Read Configuration Space\n");
}
}
// Example of using interrupts
// Example of using interrupts
printf ("\nSending Host Commands: clear HF3, clear HF4, clear HF5\n");
DSP563xx_WriteHI32Register(DSP56301 , HCVR_OFFSET, 0xF3);
DSP563xx_WriteHI32Register(DSP56301 , HCVR_OFFSET, 0xF7);
DSP563xx_WriteHI32Register(DSP56301 , HCVR_OFFSET, 0xFB);
printf ("Sending Host Commands: set HF3, set HF4\n");
DSP563xx_WriteHI32Register(DSP56301 , HCVR_OFFSET, 0xF1);
DSP563xx_WriteHI32Register(DSP56301 , HCVR_OFFSET, 0xF5);
printf ("Sending Host Command : assert INTA\n");
DSP563xx_WriteHI32Register(DSP56301 , HCVR_OFFSET, 0xFD);
printf ("\nWaiting for notification from VxD\n");
// dwReturnValue = DSP563xx_WaitForInterrupt(DSP56301, 10); //INFINITE);
dwReturnValue = DSP563xx_WaitForInterrupt(DSP56301, INFINITE);
if (dwReturnValue == WAIT_TIMEOUT) {
    printf ("\nInterrupted Timed Out\n");
}
else {
    printf ("\nReceived Interrupt: %02lx\n", dwReturnValue);
}
// Example of DSP563xx_GetPhysAdd
// Example of DSP563xx_GetPhysAdd
printf("\nExample of DSP563xx_GetPhysAdd:\n\n\tGetting the physical mem-
ory base address of HI32:");
printf ("%08lx\n", DSP563xx_GetPhysAdd(DSP56301->HI32BaseAddress,
DSP56301));
// Example of DSP563xx_LockMemoryPage and DSP563xx_UnLockMemoryPage
// Example of DSP563xx_LockMemoryPage and DSP563xx_UnLockMemoryPage
printf("\nExample of DSP563xx_LockMemoryPage and
DSP563xx_UnLockMemoryPage:\n\n");
LinearAddress = (DWORD*)malloc(0x401*sizeof(DWORD));
printf("Linear Address: %08lx", LinearAddress);
printf("\tLocking:");
if(DSP563xx_LockMemoryPage(LinearAddress, DSP56301,
0x400*sizeof(DWORD))) {
    printf(" Passed\n");
    dwReturnValue= DSP563xx_GetPhysAdd((DWORD)LinearAddress,
DSP56301);
    printf ("Physical Address: %08lx", dwReturnValue);
    printf("\tUnlocking:");

```

```

        if(DSP563xx_UnLockMemoryPage(LinearAddress, DSP56301,
0x400*sizeof(DWORD))) {
            printf(" Passed\n");
        }
        else {
            printf(" Failed\n");
        }
    }
    else {
        printf(" Failed\n");
    }
    free(LinearAddress);
    //////////////////////////////////////
    // Example of DSP563xx_UnLoadVxD
    //////////////////////////////////////
    printf("\nUnloading VxD:");
    bReturnValue = DSP563xx_UnLoadVxD(DSP56301);
    if (bReturnValue) {
        printf(" Passed\n");
    }
    else {
        printf(" Failed\n");
    }
    printf ("\nPress <Space> to exit.\n");
    while (!kbhit()){};
    _getch();

printf("\n*****\n");
////////////////////////////////////
// end of program
////////////////////////////////////
    free(DSP56301);
    return 0;
}

```

B.5 DSP56301 Sample - C-Header file

```

//-----
//
//          COPYRIGHT 1998,1999 - MOTOROLA INC.
//
//-----

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
#include <time.h>

#define      HI32_USER_MESSAGE      1
#define      DEVNODE_NOT_FOUND     0x00000010

#define      INITIALIZATION_MESSAGE 1

```

```
#define LOCK_ONE_MEMORY_PAGE 2
#define RD_CNFG_SPACE_MESSAGE 3
#define WR_CNFG_SPACE_MESSAGE 4
#define GET_PHYS_ADD_MESSAGE 6
#define UNLOCK_ONE_MEMORY_PAGE 7
```

```
#define HCTR_OFFSET 0x00000004
#define HSTR_OFFSET 0x00000005
#define HCVR_OFFSET 0x00000006
#define HRXS_OFFSET 0x00000007
#define HTXR_OFFSET 0x00000100
#define HCTR_HF0 0x00000008
#define HCTR_HF1 0x00000010
#define HCTR_HF2 0x00000020
#define HCTR_HTF0 0x00000100
#define HCTR_HTF1 0x00000200
#define HCTR_HRF0 0x00000800
#define HCTR_HRF1 0x00001000
#define HSTR_HRRQ 0x00000004
```

```
#define HI32_24BIT_MODE 0x00400000
#define HI32_32BIT_MODE 0x00000000
```

```
typedef struct
```

```
{
    char* VxDFileName;
    HANDLE DeviceHandle;
    HANDLE Evnt0;
    HANDLE Evnt3;
    char* DSPName;
    DWORD DeviceId;
    DWORD VendorId;
    DWORD HI32BaseAddress;
```

```
} DSPINFO , *pDSPINFO;
```

```
// main functions
```

```
BOOL DSP563xx_LoadVxD(pDSPINFO DSP563xx);
BOOL DSP563xx_UnLoadVxD(pDSPINFO DSP563xx);
BOOL DSP563xx_InitializeDevice(pDSPINFO DSP563xx);
DWORD DSP563xx_ReadHI32Register(pDSPINFO DSP563xx , DWORD Register);
void DSP563xx_WriteHI32Register(pDSPINFO DSP563xx , DWORD Register, DWORD
NewValue);
BOOL DSP563xx_DownloadCode(pDSPINFO DSP563xx , const CHAR *Filename);
DWORD DSP563xx_WaitForInterrupt(pDSPINFO DSP563xx , DWORD Timeout);
BOOL DSP563xx_LockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size);
BOOL DSP563xx_UnLockMemoryPage(DWORD* LinearAddress , pDSPINFO DSP563xx, DWORD
Size);
void DSP563xx_ChangeDataMode(pDSPINFO DSP563xx , DWORD HI32DataMode);
BOOL DSP563xx_ReadCfgSpace(DWORD CfgOffset , pDSPINFO DSP563xx, DWORD* Cfg-
Word);
BOOL DSP563xx_WriteCfgSpace(DWORD CfgOffset , pDSPINFO DSP563xx, DWORD Cfg-
Word);
```

```
DWORD DSP563xx_GetPhysAdd(DWORD LinAd, pDSPINFO DSP563xx);  
// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions of this function)  
HANDLE (WINAPI *GetAddressOfOpenVxDHandle())(HANDLE);  
// COPYRIGHT 1994,1995 Vireos Software, Inc. (portions of this function)  
BOOL DSP563xx_CreateCommonEvent(HANDLE* Evnt3, HANDLE* Evnt0);
```

OnCE and Mfax are registered trademarks of Motorola, Inc. Microsoft®, Visual C++®, and Windows 95/98® are registered trademarks of Microsoft Corporation. VtoolsD is a trademark of Vireo Software, Inc.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

How to reach us:

USA/Europe/Locations Not Listed:

Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1 (800) 441-2447
1 (303) 675-2140

Motorola Fax Back System (Mfax™):

TOUCHTONE (602) 244-6609
1 (800) 774-1848
RMFAX0@email.sps.mot.com

Asia/Pacific:

Motorola Semiconductors H.K. Ltd.
8B Tai Ping Industrial Park
51 Ting Kok Road
Tai Po, N.T., Hong Kong
852-26629298

Technical Resource Center:

1 (800) 521-6274

DSP Helpline

dsphelp@dsp.sps.mot.com

Japan:

Nippon Motorola Ltd
SPD, Strategic Planning Office141
4-32-1, Nishi-Gotanda
Shinagawa-ku, Japan
81-3-5487-8488

Internet:

<http://www.motorola-dsp.com/>



MOTOROLA
