# Using DMA on Trizeps4

**Documentation 1.1**

**This documents describes how to use the DMA-Ressources of Trizeps4 under WinCE.**

## 1.0 Introduction

The PXA27x processor has 32 dma channels. These may be used to transfer data between memory or from/to the on-chip peripherals. How to configure or start a dma-transfer is not in the scope of this document. Refer to the PXA27x-Processor-Family Developer-Manual for details.

Before using one of the free dma-channels, they must be assigned to a free system-interrupt-value. This is done through an interrupt-request-mechanism ( ,see the application note: „Using Interrupts on Trizeps 4“). Once assigned, you may register an Interrupt-Service-Routine ( ISR) or use a simple Interrupt-Service-Thread (IST) to control the dma-channel usage. DMA-channles 1,4 and 5 are reserved by the audio-driver.

A sample named „ssp_dma_sample“ is availlable for download.

The dma-channel functionality will be availlable as of BSP_TR4CONXS_2006Q1.

## 2.0 Getting a free DMA-Channel

As already mentioned, you may get a free dma-channel through the interrupt-request-mechanism. Each dma-channel has an individual IRQ-number. The IRQ-number consist of the number of the dma-channel stored at the lower 8bits or'd with a flag (0x03000000), which identifies this IRQ to be a dma-irq.

**TABLE 1.** DMA IRQ's

| IRQ | **Timer** |
|---|---|
| 0x03000000 | channel 0 ( reserved) |
| 0x03000001 | channel 1 ( reserved, audio-driver) |
| 0x03000002 | channel 2 |
| 0x03000003 | channel 3 |
| 0x03000004 | channel 4 (reserved, audio-driver) |
| 0x03000005 | channel 5 (reserved, audio-driver) |
| 0x03000006 | channel 6 |
| 0x03000007 ... 31 | channel 7 ... 31 |
| 0x030000FF | Get a free DMA-channel. |

To request a dma-channel you must make a call to the kernel-ioctl: IOCTL_HAL_REQUEST_SYSINTR.

Example:

```
DWORD dwIrq      = <IRQ-Number of the dma-channel to use>;
DWORD dwSysIntr = <Variable receiving the mapped SysIntr-Value; set to 0
before calling KernelIoControl>;
KernelIoControl(  IOCTL_HAL_REQUEST_SYSINTR,
                  &dwIrq,
                  sizeof(DWORD),
                  &dwSysIntr,
                  sizeof(DWORD),
                  NULL);
```

If the wanted channel has already been allocated by another application, the above KernelIoControl(..) will return FALSE.

If you use 0x020000FF to get a free dma-channel, you should also make a call to the kernel-ioctl: IOCTL_HAL_TRANSLATE_SYSINTR, to retrieve its number.

Example:

```
KernelIoControl( IOCTL_HAL_TRANSLATE_SYSINTR,
            &dwSysIntr,
            sizeof(DWORD),
            &dwIrq,
            sizeof(DWORD),
            NULL);
```

This time dwSysIntr is the input-value and dwIrq is the output-value, which will receive the dma-number (or'd with 0x03000000).

## 3.0 Assigning multiple DMA-Channels to one SysIntr

Probably the main reason to use DMA is, to send or receive data from an on-chip peripheral. Problem: you will need multiple channels to service one peripheral (i.e. SSP: one receive- and one transmit-channel). Because you will want only one Interrupt-Service-Thread to react on all DMA-Interrupts of this peripheral, you have to map these channels to one SysIntr.

This is done through the kernel-ioctl IOCTL_HAL_REQUEST_SYSINTR. This is the same function used in the chapter before. But this time, dwSysIntr is also an input-value, holding the System-Interrupt-Number.

Example:

```
DWORD dwIrq      = <IRQ-Number of the dma-channel to use>;
DWORD dwSysIntr = <SysIntr-Number to use>;
KernelIoControl(   IOCTL_HAL_REQUEST_SYSINTR,
                   &dwIrq,
                   sizeof(DWORD),
                   &dwSysIntr,
                   sizeof(DWORD),
                   NULL);
```

When using this functionality, you must know the dma-channel to use in advance, because IOCTL_HAL_TRANSLATE_SYSINTR will only return one dma-channel ( the lowest dma-channel assigned to the SysIntr-Value).

The basic procedure would be:

1. Use IOCTL_HAL_REQUEST_SYSINTR with a given dma-channel and dwSysIntr set to 0. After calling this kernel-ioctl, dwSysIntr holds the assigned system-interrupt-number.
2. Use IOCTL_HAL_REQUEST_SYSINTR with the next dma-channel and dwSysIntr-value from step 1.

## 4.0 What to do on Interrupts

This depends on what approach you have chosen to use, ISR or IST.

### 4.1 What to do, if you use an IST

If you use an IST, the kernel takes care of disabling the global DMA interrupt. After this is done, it signals the System-Interrupt previously assigned with IOCTL_HAL_REQUEST_SYSINTR.

In the IST you have to clear the interrupt cause ( see DCSR-registers of the dma-controller) and reenable the DMA-interrupt through InterruptDone( dwSysIntr).

**Note:**
It's very important to reenable the global DMA-interrupt through InterruptDone, otherwise other drivers (i.e. audio-driver) might stop functioning correctly.

### 4.2 What to do, if you use an ISR

If you use an ISR, the kernel calls your Interrupt-Service-Routine directly (,with little overhead of code) after the interrupt occured.

The kernel does not disable the global DMA-interrupt. And you should not either.

Procedure to be done in the ISR:

1. Check the interrupt cause in the DCSR-register.
2. Do your processing.
3. Clear the interrupt-cause.

## 5.0 Cleaning up

If you don't need the dma-channels anymore, you should free the dma-ressource, so that other applications may use it.

To do this, make a call to InterruptDisable(..), which is used to disable the dma-interrupt. This function is currently a stub, but could be used in future. After that call the kernel-ioctl IOCTL_HAL_RELEASE_SYSINTR to release the ressource. That will release all dma-channels assigned to the system-interrupt-value.

Example:

```
DWORD dwSysIntr = <Variable containing SysIntr-Value, received through
IOCTL_HAL_REQUEST_SYSINTR>;

InterruptDisable( dwSysIntr);
KernelIoControl(  IOCTL_HAL_RELEASE_SYSINTR,
                  &dwSysIntr,
                  sizeof(DWORD),
                  NULL,
                  0,
                  NULL);
```