Application Note for ConnectCore module accessing external perhiperals

Digi International Inc.

11001 Bren Road East

Minnetonka, MN 55343 (USA)

☎ +1 877 912-3444 or +1 952 912-3444

http://www.digi.com

Table of contents

History

Date	Version	Author	Description
03/12/07	0.1	ME	First version

References

Number	Name	Description
1		
2		

Acronyms

Name	Description
OS	Operating System.
CS	Chip Select
PB	Platform Builder
BSP	Board Support Package
OAL	Original Equipment Manufacturer
IST	Interrupt Service Thread
ISR	Interrupt Service Routine
WE	Write Enable
DLL	Dynamic Link Library
DM	Device Manager
PM	Power Manager

4 Overview

4.1 Introduction

This document will describe what is necessary to access a device connected to the peripheral external bus of the NS9360 or ConnectCore 9 modules.

External peripherals are normally accessed through an address mapping and the corresponding read and write cycles are handled by the OE# and WR# signal that are available on the external bus.

The CS signal is used to map the connected device on the external bus to a specific memory address range. If several devices will be connected to the same CS signal address space additional logic might be necessary to access each of the connected devices.

These control signals have the possibility to adjust to timing configuration of the peripheral device that is connected to the bus. The NS9360 CPU contains several control register to adjust the external bus timing of the CS#, WR# and OE# signal. This allows connecting a huge variety of devices to the external bus to increase or enhance the functionality of the CPU. Additionally, the CS signal and the address range are configurable by a register of the CPU. In that register the start of the CS address and the CS mask can be setup.

The bus width to access to an external peripheral can be 8bit, 16bit or 32bit. In the static memory configuration register that value can be set to select one of the three bus width.

For more details please refer to the hardware manual of the corresponding CPU.

5 External Peripheral Implementation

5.2 Device Driver example

The access to the external peripheral can be done through a Windows CE standard stream interface device driver for more details on that kinds of driver please refer the online help. Depending on the external peripheral also another driver models could be used.

The stream interface driver can be implemented either in kernel or user mode depending on the performance that is required to access the connected device.

5.2.1 Stream Interface Driver Functions

A Windows CE driver is normally implemented as a DLL. The DLL will exports functions that allow applications to access the driver. The main entry point into a DLL is DllEntry. That is called when the DLL is loaded either by the DM or an application.

The standard stream interface driver uses normally the following functions.

Driver Function	API call	Description
XXX_Init()		Called by the DM when loaded
XXX_DeInit()		Called when unloaded
XXX_Open()	CreateFile()	Opens a driver handle
XXX_Close()	CloseHandle()	Closes a driver handle
XXX_Read()	ReadFile()	Read from the external device
XXX_Write()	WriteFile()	Write to the external peripheral
XXX_PowerUp()		Called by the PM
XXX_PowerDown()		Called by the PM
XXX_Seek()	SetFilePointer()	Set data pointer
XXX_IOControl()	DeviceIoControl	Driver specific IOCTLs

Table 5.2.1.1: Stream interface functions

The XXX of each function need to be changed by a three letter prefix that will be used to identify each of the functions implemented in the driver.

5.2.2 Export Driver functions

The functions that are exported by the driver are listed in a file with the extension DEF.

5.2.3 Registry settings

To load the driver by the DM, for more information of the DM task please refer to the online help, a specific registry path need to be used. The DM checks all entries under the path *HKEY_LOCAL_MACHINE\Drivers\BuiltIn* and tries to load each of the driver entries.

Following an example of such a registry entry for a Windows CE driver

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\My_Driver_Name]
  "DeviceArrayIndex"=dword:0
  "Prefix"="XXX"
  "Dll"="my_driver_name.dll"
  "Order"=dword:0
   "MyRegKey"=dword:ffff
```

All DWORD values in the registry are red as hexadecimal and all strings as Unicode by the OS.

5.2.3 Driver API calls

To access the driver from an application there are only a few APIs available for a stream interface driver. Table 5.2.1.1 shows each of the APIs that are available, for more details on each of the APIs and the necessary parameters please refer to the Platform Builder online help.

5.3 Low Level functions

The BSP for the ConnectCore 9 family includes several low level functions that can be used to configure the external bus interface of the NS97XX CPU family. These functions can be found in the BSP sources at

%(_WINCEROOT)\platform\common\src\soc\NS9XXX_DIGI_V1\oal\oalmemcrtl

Here a list of the current functions implemented in the BSP and their functionality

Low level function	Description
OEMNS9xxx_set_static_mem_cfg	Writes static memory config register
OEMNS9xxx_get_static_mem_cfg	Read static memory config register
OEMNS9xxx_set_static_mem_wen	Set write enable delay time
OEMNS9xxx_set_static_mem_oen	Set output enable delay time
OEMNS9xxx_set_static_mem_rd	Set read delay time
OEMNS9xxx_set_static_mem_pg	Set page mode read delay time
OEMNS9xxx_set_static_mem_wr	Set write delay time
OEMNS9xxx_set_static_mem_tn	Set turn round delay time
OEMNS9xxx_set_static_mem_extw	Set extended wait state

Table 5.3.1: Low level BSP functions

5.4 Use Interrupt to Access External Device

To reduce CPU load and increase performance to access the external peripheral device an interrupt should be used. The NS9XXX series has 4 external interrupt lines that can be used to implement that functionality into the system. Please refer to the hardware manual to verify which external interrupt line is free for use.

There are two possibilities to implement the interrupt functionality to the OS.

1. Implement an IST in the device driver of the external peripheral device. When an interrupt occurs the system will enter the IST and perform the necessary operations

2. Use from the application the GPIO driver to detect an incoming interrupt. The GPIO driver included in the BSP includes several IOCTLs to configure a pin and to detect an interrupt on the four external interrupt pins. For more information on how to use the GPIO driver please review the “Windows CE User Guide”.

5.4.1 Implement IST in Driver

The implementation of the interrupt into the driver is normally made by using a thread that handles the interrupt and an event that is linked to the interrupt itself. In the thread the driver waits that the OS signals that the created event has be signaled.

Windows CE offers the possibility to add an installable ISR to the system to avoid modifying the OAL to implement the interrupt, because in some cases it might not be possible to modify the OAL.

The Platform Builder already comes with a generic installable ISR. If that installable ISR fulfills the needs for the driver the following entries need to be added to the driver registry.

```
"IsrDll"="giisr.dll"
"IsrHandler"="ISRHandler"
```

In case the installable ISR need to be modified the source code is available at

%(_WINCEROOT)\PUBLIC\COMMON\OAK\DRIVERS\GIISR

5.4.2 Use GPIO Driver

The GPIO driver can be accessed from any Windows CE application. A source code example on how to use the GPIO driver is included in the BSP.

How to configure and use an external IRQ line with the GPIO driver the following code example

```
….
GPIOMessage gpioMessage;
DWORD dwMessagesTransferred;

/* opening the GPIO driver and receive a handle */

gpioMessage.unPinNumber = unPinNumber;      // external irq pin
gpioMessage.mode = nMode;

gpioMessage.ulFlags = 0;
gpioMessage.unValue = 0;

/* wait for the external interrupt to occur */
if (!DeviceIoControl(hPort, IOCTL_GPIO_WAIT_FOR_IRQ, &gpioMessage,
    sizeof(gpioMessage), 0, 0, &dwMessagesTransferred, 0))
{
    /* failure while waiting for interrupt */
}
else
{
    /* received external interrupt */
}
```

When an interrupt occurs on the external interrupt line the driver will signal that to the application returning from API. Now the application can do the necessary steps to access the external device.

5.5 Static Memory Register

The following table shows all the static memory register available on the NS9XXX family.

Register	Default	Max. Value	Min. Value	Description
Extended Wait	0	0x3FF	0x001	Long static write and read
Configuration	0	0x18X1CF	0x000000	Static memory configuration
Write Enable Delay	0	0x0F	0x01	Delay from CS to WE
Output Enable Delay	0	0x0F	0x01	Delay from CS to OE
Read Delay	0x1F	0x1F	0x00	Delay from CS to read access
Page Mode Read Delay	0x1F	0x1F	0x00	Delay asynchronous page mode access
Write Delay	0x1F	0x1F	0x00	Delay from CD to write access
Turn Round Delay	0x0F	0x0F	0x00	Number of bus turnaround cycles

Table 5.5.1: Static Memory Register of NS9XXX family

For more details on each of the register, please refer the hardware manual of the corresponding CPU.