Mini6410 GPIO Driver

Version 1.0 : 2014/02/03 : D. Valois : Creation.

The delivery is a zip file that contains :
- the binary of the driver : **gpio.dll**
- the definition file **gpio.h**
- a directory **C++ sample** that contains a C++ sample application that uses the driver
- a directory **VB.Net sample** that contains a VB.Net sample application that uses the driver

The trial version of the driver is free but is limited : you can change only 50 times the outputs, and read 5000 times the state of the inputs.

If you want to acquire the full version of this driver, please contact me via my email : dominique.valois@gmail.com

1. Integration of the driver in your Windows Embedded CE image

In Windows Embedded CE, the system functions that allow access to the hardware are executed in kernel mode. Because of that, an application can not directly access to the hardware, and the driver must be included in the Windows Embedded image during its generation.

If you don't want to generate your own Windows Embedded CE image, I can do it for you. Don't hesitate to contact me via my email : dominique.valois@gmail.com

In the following paragraph, I consider that you have chosen the default directory to store the Windows Embedded CE source code (c:\wince600) . If you choose a different directory, you must adapt the directory path.

To integrate the driver in your Windows Embedded CE image, you must follow these steps :

Step 1 :
Copy the file **gpio.dll** in **C:\WINCE600\PLATFORM\SMDK6410\FILES**

Step 2 :
Copy the file **gpio.h** in **C:\WINCE600\PLATFORM\SMDK6410\SRC\INC**

Step 3 :
Add the following line at the end of the file **C:\WINCE600\PLATFORM\SMDK6410\FILES\platform.bib :**

```
gpio.dll $(_FLATRELEASEDIR)\gpio.dll NK SHK
```

Step 4 :
Add the following lines at the end of file **C:\WINCE600\PLATFORM\SMDK6410\FILES\platform.reg :**

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\GPI]
     "Prefix"="GPI"
     "Dll"="GPIO.DLL"
     "Order"=dword:100
     "Index"=dword:0
     "FriendlyName"="Domodom GPIO Driver"
```

Then you can generate and flash the new nk.bin in your mini6410 board. If everything works fine and if you activated seria debug traces, following message must be displayed at startup :

```
Chargement du driver ADC...
```

2. Use of the driver

The driver is a stream driver that receives commands from applications by means of file system calls.

The Createfile function opens the driver and returns a handle to the driver.

The handle allow to send commands to the driver with the DeviceIoControl() function.

The implemented commands are detailed in the following paragraphs.

2.1 Implemented commands

1 IOCTL_GPIO_SET_PIN_CONFIGURATION

This command must be used to configure a pin :
- as an output
- as an input with pullup
- as an input without pullup

Be careful, some pins can not be configured as input or output, or don't have integrated pullup, etc… The driver doesn't' verify the configuration asked by the user !

The data sent with this command is a buffer of 4 bytes :
Byte 0 : port number -> 0 for port A, 1 for port B, etc…
Byte 1 : pin number in the port
Byte 2 : pin configuration -> 0 for output, 1 for input with pullup, 2 for input without pullup
Byte 3 : value of the pin if configured as an output (0 for ON, 1 for OFF)

2 IOCTL_GPIO_SET_PIN_OUTPUT_VALUE

This command is used to configure the state of a pin previously configured as an output.

The data sent with this command is a buffer of 3 bytes :
Byte 0 : port number -> 0 for port A, 1 for port B, etc…
Byte 1 : pin number in the port
Byte 2 : value of the output (0 for ON, 1 for OFF)

3 IOCTL_GPIO_GET_PIN_INPUT_VALUE

This command is used to read the state of a pin previously configured as an input.

The data sent with this command is a buffer of 3 bytes :
Byte 0 : port number -> 0 for port A, 1 for port B, etc…
Byte 1 : pin number in the port
Byte 2 : this value is not used by the driver

After the execution of this command, the state of the input is copied in the byte 2 of the buffer.

4 IOCTL_GPIO_GET_VERSION

This command is used to read the version of the driver.

The data send must be a buffer of 10 bytes in which the version number will be copied as a string.

3. C++ sample of use

This method must be preferred if you need an application with good performances because it doesn't use managed code.

3.1 Open the device

```
HANDLE hGPIO;

hGPIO = CreateFile(L"GPI0:", GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
      FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, 0);
```

3.2 Configure a pin as an output

```
GPIO_SET_PIN_CONFIGURATION    gpioInit;

gpioInit.portNumber           = PORT_B;
gpioInit.pinConfiguration     = OUTPUT;
gpioInit.pinValue             = OFF;
gpioInit.pinNumber            = 5

DeviceIoControl(hGPIO, IOCTL_GPIO_SET_PIN_CONFIGURATION, &gpioInit,
      sizeof(GPIO_SET_PIN_CONFIGURATION), NULL, 0, 0, NULL);
```

The above code configures the pin 5 of the port B in output and set the output to OFF (low level).

3.3 Configure a pin as an input

```
GPIO_SET_PIN_CONFIGURATION    gpioInit;

gpioInit.portNumber           = PORT_G;
gpioInit.pinConfiguration     = INPUT_WITH_PULLUP;
gpioInit.pinNumber            = 0;

DeviceIoControl(hGPIO, IOCTL_GPIO_SET_PIN_CONFIGURATION, &gpioInit,
      sizeof(GPIO_SET_PIN_CONFIGURATION), NULL, 0, 0, NULL);
```

The above code configures the pin 0 of the port G in input with pullup.
If you don't want to use the internal pullup, you can select **INPUT_WITHOUT_PULLUP_WITHOUT_PULLDOWN** instead of **INPUT_WITH_PULLUP.**
If you want to use the internal pulldown, you can select **INPUT_WITHOUT_PULLDOWN** instead of **INPUT_WITH_PULLUP.**

3.4 Configure the state of an output

```
GPIO_SET_PIN_OUTPUT_VALUE     gpioPinValue;

gpioPinValue.portNumber       = PORT_B;
gpioPinValue.pinNumber  = 5;
gpioPinValue.pinValue   = ON;

DeviceIoControl(hGPIO, IOCTL_GPIO_SET_PIN_OUTPUT_VALUE, &gpioPinValue,
sizeof(GPIO_SET_PIN_OUTPUT_VALUE), NULL, 0, 0, NULL);
```

The above code set an output to **ON** (high level). For the low level, use **OFF**.

3.5 Read the state of an input

```
GPIO_GET_PIN_INPUT_VALUE      gpioPinValue;
```

```
gpioPinValue.portNumber       = PORT_G;
gpioPinValue.pinNumber        = 0;

DeviceIoControl(hGPIO, IOCTL_GPIO_GET_PIN_INPUT_VALUE, &gpioPinValue,
      sizeof(GPIO_GET_PIN_INPUT_VALUE), NULL, 0, 0, NULL)
```

The above code reads the state of an input. The state of the input is store in **gpioPinValue.pinValue**.

3.6 Close the device

```
CloseHandle(hGPIO);
```

The driver must be closed before the end of the application.

4. VB.Net sample of use

This method uses managed code, but a driver runs in the kernel and its code is not managed, then the system has to switch between managed and unmanaged code/data. These switch operations are slow, then If you need to design an application that runs very fast, I encourage you to write the application in C/C++.

4.1 Code of the commands

```
Private Const IOCTL_GPIO_SET_PIN_CONFIGURATION As Integer = 262144
Private Const IOCTL_GPIO_SET_PIN_OUTPUT_VALUE As Integer = 262148
Private Const IOCTL_GPIO_GET_PIN_INPUT_VALUE As Integer = 262152
```

4.2 Constants

The following constants must be declared :

```
Friend Enum GPIO_PORT As Integer
    PORT_A = 0
    PORT_B = 1
    PORT_C = 2
    PORT_D = 3
    PORT_E = 4
    PORT_F = 5
    PORT_G = 6
    PORT_H = 7
    PORT_I = 8
    PORT_J = 9
    PORT_K = 9
    PORT_L = 9
    PORT_M = 9
End Enum

Friend Enum GPIO_PIN_CONFIGURATION As Integer
    OUTPUT = 0
    INPUT_WITH_PULLUP = 1
    INPUT_WITHOUT_PULLUP_WITHOUT_PULLDOWN = 2
    INPUT_WITHOUT_PULLDOWN = 3
End Enum

Friend Enum GPIO_PIN_VALUE As Integer
    LOW = 0
    HIGH = 1
End Enum
```

4.3 Data structures definition

The data exchanged with the driver are defined like this :

```
<StructLayout(LayoutKind.Explicit)> _
   Friend Structure GPIO_SET_PIN_CONFIGURATION
    <FieldOffset(0)> Public portNumber As GPIO_PORT
    <FieldOffset(4)> Public pinNumber As Integer
    <FieldOffset(8)> Public pinConfiguration As GPIO_PIN_CONFIGURATION
    <FieldOffset(12)> Public pinValue As GPIO_PIN_VALUE
End Structure

<StructLayout(LayoutKind.Explicit)> _
   Friend Structure GPIO_SET_PIN_OUTPUT_VALUE
    <FieldOffset(0)> Public portNumber As GPIO_PORT
    <FieldOffset(4)> Public pinNumber As Integer
```

```
        <FieldOffset(8)> Public pinValue As GPIO_PIN_VALUE
    End Structure

    <StructLayout(LayoutKind.Explicit)> _
       Friend Structure GPIO_GET_PIN_OUTPUT_VALUE
        <FieldOffset(0)> Public portNumber As GPIO_PORT
        <FieldOffset(4)> Public pinNumber As Integer
        <FieldOffset(8)> Public pinValue As GPIO_PIN_VALUE
    End Structure
```

4.4 Import the Createfile and DeviceIOControl fonctions

```
<DllImport("coredll.dll")> _
    Public Shared Function DeviceIoControl(ByVal hDevice As Integer, ByVal
dwIoControlCode As Integer, ByVal lpInBuffer As IntPtr, ByVal nInBufferSize As +
+Integer, ByVal lpOutBuffer As Byte(), ByVal nOutBufferSize As Integer, ByRef
lpBytesReturned As Integer, ByVal lpOverlapped As IntPtr) As Integer
    End Function

<DllImport("coredll.dll")> _
    Public Shared Function CreateFile(ByVal lpFileName As String, ByVal
dwDesiredAccess As UInteger, ByVal dwShareMode As UInteger, ByVal lpSecurityAttributes
As IntPtr, ByVal dwCreationDisposition As UInteger, ByVal dwFlagsAndAttributes As
UInteger, ByVal hTemplateFile As IntPtr) As IntPtr
    End Function
```

4.5 Open the driver

```
    Private Shared _gpioFile As IntPtr = CreateFile("GPI0:", &H40000000, 0,
IntPtr.Zero, 3, 0, IntPtr.Zero)
```

4.6 Configure a pin as an output

```
' Allocate a block of unmanaged memory to receive the structure
bytePtr = Marshal.AllocHGlobal(16)

pinConfig.pinConfiguration    = GPIO_PIN_CONFIGURATION.OUTPUT
pinConfig.pinValue            = GPIO_PIN_VALUE.LOW
pinConfig.portNumber          = GPIO_PORT.PORT_B
pinConfig.pinNumber           = 5

' Copy the structure to the memory block
Marshal.StructureToPtr(pinConfig, bytePtr, False)

DeviceIoControl(_gpioFile, IOCTL_GPIO_SET_PIN_CONFIGURATION, bytePtr, 16, size, 0,
accessType, IntPtr.Zero)
```

The above code configures the pin 5 of the port B in output and set the output to OFF (low level).

4.7 Configure a pin as an input

```
' Allocate a block of unmanaged memory to receive the structure
bytePtr = Marshal.AllocHGlobal(16)

pinConfig.portNumber          = GPIO_PORT.PORT_G
pinConfig.pinConfiguration    = GPIO_PIN_CONFIGURATION.INPUT_WITH_PULLUP
pinConfig.pinNumber           = 0

' Copy the structure to the memory block
Marshal.StructureToPtr(pinConfig, bytePtr, False)

DeviceIoControl(_gpioFile, IOCTL_GPIO_SET_PIN_CONFIGURATION, bytePtr, 16, size, 0,
accessType, IntPtr.Zero)
```

The above code configures the pin 0 of the port G in input with pullup.
If you don't want to use the internal pullup, you can select **INPUT_WITHOUT_PULLUP_WITHOUT_PULLDOWN** instead of **INPUT_WITH_PULLUP.**
If you want to use the internal pulldown, you can select **INPUT_WITH_PULLDOWN** instead of **INPUT_WITH_PULLUP.**

4.8 Configure the state of an input

```
' Allocate a block of unmanaged memory to receive the structure
bytePtr = Marshal.AllocHGlobal(16)

Dim pinValue As GPIO_SET_PIN_OUTPUT_VALUE

pinValue.pinValue       = GPIO_PIN_VALUE.LOW
pinValue.portNumber     = GPIO_PORT.PORT_B
pinValue.pinNumber      = 0

' Copy the structure to the memory block
Marshal.StructureToPtr(pinValue, bytePtr, False)

DeviceIoControl(_gpioFile, IOCTL_GPIO_SET_PIN_OUTPUT_VALUE, bytePtr, 16, size, 0,
accessType, IntPtr.Zero)
```

The above code set an output to **ON** (high level). For the low level, use **OFF**.

4.9 Read the state of an output

```
Dim pinValueRead As GPIO_GET_PIN_OUTPUT_VALUE

' Allocate a block of unmanaged memory to receive the structure
bytePtr = Marshal.AllocHGlobal(16)

pinValue.portNumber = GPIO_PORT.PORT_G
pinValue.pinNumber = 5

' Copy the structure to the memory block
Marshal.StructureToPtr(pinValue, bytePtr, False)
DeviceIoControl(_gpioFile, IOCTL_GPIO_GET_PIN_INPUT_VALUE, bytePtr, 12, Size, 0,
accessType, IntPtr.Zero)

' copy the memory block to the structure
pinValueRead = CType(Marshal.PtrToStructure(bytePtr,
GetType(GPIO_GET_PIN_OUTPUT_VALUE)), GPIO_GET_PIN_OUTPUT_VALUE)
```

The above code reads the state of an input. The state of the input is store in `pinValueRead.pinValue`.

Be careful, the driver doesn't verify that the pin has been configured as an input !