Phytec Windows CE 6.0
i.MX31 Evaluation Kit BSP
Documentation

Document Version	0.7
BSP Version	0.94
April 2008	

Table of Contents

1. BSP Contents

The BSP Release contains the following folders and files:

- **`WINCE600:`**

  contains the Source and Build files needed to build an Windows CE 6.0 Image for the Phytec i.MX31 Evaluation Kit. For installation instructions, refer to chapter 3.

- **`Example_Project:`**

  contains an example project to build an Image utilizing nearly all supported features. Again, refer to chapter 3 for installation instructions.

- **`Documentation:`**

  contains documentation related to the BSP.

- **`Splashscreen:`**

  contains an example bitmap to use as splashscreen.

Folder structure of the BSP:

```
PLATFORM
|--COMMON
|  |--SRC
|     |--ARM
|     |  |--FREESCALE
|     |     |--MX31
|     |     |--MXARM11
|     |     |--PMIC
|     |
|     |--SOC
|        |--FREESCALE
|           |--COMMON
|           |--MX31
|           |--MXARM11
|           |--PMIC
|
|--iMX31_Phytec
   |--CATALOG
   |--FILES
   |--SRC
      |--BOOTLOADER
      |--COMMON
      |  |--ARGS
      |  |--BSPCMN
      |  |--NORCFI
      |  |--PERREG
      |  |--SMSC9118
      |  |--ST_NANDFMD
      |
      |--DRIVERS
      |--KITL
      |--OAL
```

2. Requirements

Microsoft Windows Embedded CE 6.0

- Windows CE 6.0 Updates:
  - Windows Embedded CE 6.0_Product_Update_Rollup_2006
  - Windows Embedded CE 6.0_Update_070131_2007M01
  - Windows Embedded CE 6.0_Update_070228_2007M02
  - Windows Embedded CE 6.0_Update_070331_2007M03
  - Windows Embedded CE 6.0_Update_070430_2007M04
  - Windows Embedded CE 6.0_Update_070630_2007M06
  - Windows Embedded CE 6.0_Update_070731_2007M07
  - Windows Embedded CE 6.0_Update_070831_2007M08
  - Windows Embedded CE 6.0_Update_070930_2007M09
  - Windows Embedded CE 6.0 R2_Update_071130_2007M11
  - Windows Embedded CE 6.0 R2_Update_071231_2007M12
  - Windows Embedded CE 6.0 R2_Update_Rollup_2007

  , all available on the Microsoft Website.

3. Installation and Build - Instruction

3.1. Preparation

Before you install the BSP, be shure to perform the following steps:

In order to avoid problems, it is recommended to remove any previously installed i.MX31 BSPs. To do so, please refer to the uninstall instructions shipped with the corresponding BSP.

If you decide to not remove an installed i.MX31 BSP, be warned that the installation if the Phytec i.MX31 BSP will override all changes you made to files contained in this BSP. So either save your changed files to another location or consider removing the 'old' BSP.

3.2. BSP Installation

The installation step consists of simply copying the folders contained in the 'WINCE600' folder into the existing 'WINCE600' folder on your harddrive.

> ***CAUTION: as mentioned above, this step will override any files associated with a previously installed i.MX31 BSP!***

To avoid problems building an image, inspect the SOURCES - file located under 'WINCE600/PLATFORM/COMMON/SRC/ARM' and ensure that 'FREESCALE' is under 'DIRS_ARM' rather than under 'OPTIONAL_DIRS'.

3.3. Example Project Installation

If you want to use the provided example project, copy the entire 'Example_Project' folder into the 'OSDesigns' folder on your harddrive.

3.4. Building an Image

The best way to start building an image for the Phytec i.MX31 platform is to use the provided example project. To do so, open platform builder and perform File -> Open -> Project/Solution and then choose the *.pbxml file from the example project directory. After that, choose Build -> Build Solution.

4. Supported Features

The following table lists all features supported by this release of the BSP.

Feature	supported?	Comment
OAL		
KITL	yes	at the moment, the used KITL-Connection is fixed to 'active KITL'
serial debug	yes	serial debug messages show up on UART3
I^2C	yes	used to read and write to the external RTC on I^2C2
external RTC	yes	• used to save the time during power-off • MC13782 RTC is synchronized to external RTC at startup • external RTC is synchronized, when time is changed in controlpanel
MC13783 RTC	yes	used for WinCE time-of-day
Splashscreen	yes	display of a custom bitmap on the display on startup. Only available for Sharp LQ035Q7DH06 panel.
Drivers		
Display	yes	Support for: • Sharp LQ035Q7DH06 (QVGA) • Sharp LQ057Q3DC12 (QVGA) • LG LB121S02 (800x600)
Camera	no	
CSPI 1	yes	Bus Driver for CSPI interface
Serial	yes	UART1 for serial communication
GPT	yes	
PWM	yes	
NANDFC	yes	currently only filesystem available, boot from NAND not verified yet
One Wire	yes	tested with DS1921 temp - sensor
Audio	yes	Playback and Recording via Line-In
MMC/SD	partial	• MMC Memory Cards work • Problems with some SD Cards
PCMCIA	yes	Problems with some cards, hardware problem is under investigation.
USB Host	yes	USB HighSpeed Host Port for Mass-Storage and HID (Mouse, Keyboard)
USB OTG	yes	OTG Port to establish an ActiveSync connection to PC

I^2C	yes	Bus Driver for e.g. Camera and EEPROM
Ethernet	yes	• Download Image over Ethernet in Eboot • WinCE NDIS Driver
CAN Controller	yes	Driver for SJA1000 CAN - Controller (Binary only)
MC13783 PMIC	yes	Used as: • Touch Controller • Audio Controller • ADC • Battery Controller and Charger
Battery	no	
EEPROM	yes	

5. Driver Documentation

5.1. Display Driver

The Display Driver is a Direct Draw Driver using the Synchronous Display Controller (SDC) of the i.MX31 Image Processing Unit (IPU). It supports the Sharp LQ035Q7DH06 QVGA panel (default panel), the Sharp LQ057Q3DC12 QVGA panel and the LG LB121S02 SVGA panel.

Summary

Driver Attribute	Definition
MXARM11 SOC Driver Path	`N/A`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\IPU\DISPLAY\DDIPU_SDC`
Import Library	`ddgpe.lib, gpe.lib`
Driver DLL	`ddraw_ipu_sdc.dll`

Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> Display -> Sharp LQ035Q7DH06 (QVGA)`
	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> Display -> Sharp LQ057Q3DC12 (QVGA)`
	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> Display -> LG LB121S02 (800x600)`

SYSGEN Dependency	`SYSGEN_DDRAW=1`
BSP Environment Variable	*`BSP_DISPLAY_SHARP_LQ035Q7DH06`* `for Sharp LQ0357DH06 QVGA panel` *`BSP_DISPLAY_SHARP_LQ057Q3DC12`* `for Sharp LQ057Q3DC12 QVGA panel` *`BSP_DISPLAY_LG_LB121S02`* `for LG LB121S02 SVGA panel`

Software Operation

Communication with the Display is done using the Graphics Device Interface (GDI) API and the DirectDraw API, both defined by Microsoft (please refer to ***Windows CE Features −> Shell and User Interface −> Graphics, Windowing and Events −> GWES Application Development −> Graphics Device Interface*** for help on GDI and to ***Windows CE Features −> Graphics and Multimedia Technologies −> Graphics −> DirectDraw for help on DirectDraw***).

The Display Driver supports the following DirectDraw features:

- page flipping (one backbuffer)
- Overlay surfaces using RGB or the FOURCC UYVY pixel format.
- Overlaying using a color key for the overlay surface for RGB colors.
- Overlaying using a color key for the non-overlay graphics surface for RGB colors.
- Stretching of overlay surfaces.

The Display Driver also uses Post-Processing features of the i.MX31 IPU to provide the following possibilities:

- Color space conversion of UYVY overlay data to RGB. This conversion is required in order to combine the overlay data with RGB graphics plane data in the IPU SDC.
- Resizing of the overlay surface.
- Rotation of the overlay surface (used when the screen orientation is rotated, see 'Button Operation' above).
- Resizing and rotation of the primary graphics surface.

In order to communicate directly with the display driver, an escape code mechanism is provided. Descriptions of standard escape codes can be found under ***Developing a Device Driver −> Windows Embedded CE Drivers −> Display Drivers −> Display Driver Escape Codes*** in the Platform Builder Help.

Configuration

The Display Driver is configured using the registry keys described below. Depending on the selected display in the catalog, the corresponding registry settings are included.

If the '*Sharp LQ035Q7DH06 (QVGA)*' panel is selected, the following registry settings are included:

```
[HKEY_LOCAL_MACHINE\Drivers\Display\DDIPU_SDC]
      "Bpp"=dword:10                ; 16bpp
      "PanelType"=dword:4           ; Sharp QVGA Panel (3,5 Inch)
      "VideoMemSize"=dword:350000
```

If the '*Sharp LQ057Q3DC12 (QVGA)*' panel is selected, the following registry keys are included:

```
[HKEY_LOCAL_MACHINE\Drivers\Display\DDIPU_SDC]
      "Bpp"=dword:10                ; 16bpp
      "PanelType"=dword:1           ; Sharp Panel
      "VideoMemSize"=dword:350000
```

If the '*LG LB121S02 (800x600)*' panel is selected, the following registry keys are included:

```
[HKEY_LOCAL_MACHINE\Drivers\Display\DDIPU_SDC]
      "Bpp"=dword:10                ; 16bpp
      "PanelType"=dword:2           ; LG Panel
      "VideoMemSize"=dword:800000
```

`VideoMemSize` determines the amount of memory used by the driver for the various surfaces.

`Bpp` sets the bytes per pixel of the display, this is set to 16-bit RGB pixel data for all supported displays.

5.2 Camera Driver

The camera sensor port is not supported yet.

5.3 SPI Driver

The SPI Driver allows rapid serial communication and can be confiured to master- or slave-mode. SPI1 SS0 can not be accessed using this driver because it is used to communicate with the MC13783 PMIC.

Summary

Driver Attribute	Definition
MXARM11 SOC Driver Path	..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\CSPIBUS
MX31 SOC Driver Path	..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MX31\DRIVERS\CSPIBUS
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\CSPIBUS
Import Library	N/A
Driver DLL	cspi.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_PHYTEC -> Device Drivers -> CSPI Bus -> CSPI 1
SYSGEN Dependency	N/A
BSP Environment Variable	BSP_CSPIBUS1=1

Software Operation

The CSPI Driver is a stream interface driver and therefore can be accessed using the filesystem API. Before working with SPI devices, a handle to the driver must be obtained using the `CreateFile` (special device `SPI1:`) function. As always, this handle has to be closed using the `CloseHandle` function, when no longer needed. Configuring the CSPI and exchanging data is done using the following IOControlCode and Structures:

IOCTLs:

CSPI_IOCTL_EXCHANGE

relevant Parameters:

lpInBuffer Pointer to a `CSPI_XCH_PKT_T` structure containing all necessary information for the transfer (see below).

nInBufferSize Size of the `CSPI_XCH_PKT_T` structure used.

Structures:

```
typedef struct
{
unsigned int data:24;
unsigned int null:1;
unsigned int address:6;
unsigned int rw:1;
} CSPI_PACKET32_FIELDS;
```

Members

reg

Object points to register value (`CSPI_PACKET32_FIELDS` structure) for a read or write operation depending on the rw value.

data

this field is an alternative to the `CSPI_PACKET32_FIELDS`, using a 32 Bit value.

```
typedef union
{
CSPI_PACKET32_FIELDS reg;
unsigned int data;
} CSPI_PACKET32;
```

Members

data

Data that is needed to write to the SPI bus.

null

Unused bit in the register. Should be null.

address

Address from which data to read or write.

rw

Field used to issue read or write command.

```
typedef struct
{
    UINT8   chipselect;
    UINT32  freq;
    UINT8   bitcount;
    BOOL    sspol;
    BOOL    ssctl;
    BOOL    pol;
    BOOL    pha;
    UINT8   drctl;
} CSPI_BUSCONFIG_T, *PCSPI_BUSCONFIG_T;
```

Members

chipselect

Selects one of four external SPI Master/Slave Devices. In master mode, these two bits select the external slave devices by asserting the SSn outputs. Only the selected SSn signal will be active while the remaining 3 signals will be negated.

freq

Frequency to which CSIP bus has to be configured.

bitcount

Number of bits to be transmitted at a time.

sspol

Selects the polarity of the chipselect signal.

ssctl

SS Signal waveform select Selects the polarity of the chipselect signal.

pol

SPI clock polarity control.

pha

SPI Clock/Data Control

drctl

SPI Data Ready Control

```
typedef struct
{
    PCSPI_BUSCONFIG_T pBusCnfg;
    LPVOID pTxBuf;
    LPVOID pRxBuf;
    UINT32 xchCnt;
    HANDLE xchEvent;
} CSPI_XCH_PKT_T, *PCSPI_XCH_PKT_T;
```

Members

pBusCnfg

Object points to the bus configuration of the SPI bus.

pTxBuf

A pointer to a buffer of bytes to transmit in a write operation.

pRxBuf

A pointer to a buffer that will be read into during a read operation.

xchCnt

The number of packets to transmit from pTxBuf / read to pRxBuf.

xchEvent

An event handle that is signalled whenever the packet operation (read or write) has completed. **CreateEvent** must be called to create the event before the packet can be passed to the CSPI.

Bus Configuration

In order to successfully communicate with a SPI device, the CSPI module first has to be configured properly. This is done by setting up a **CSPI_BUSCONFIG_T** structure according to the intended configuration. After that, this structure is set as the **pBusCnfg** -field of a **CSPI_XCH_PKT_T** structure. It is also necessary to create an event, which is passed as the **xchEvent** -field of the **CSPI_XCH_PKT_T** structure. This event will be signaled when the configuration is done.
After all this is set up, the configuration is actually done by calling `DeviceIoControl` with an control code of **CSPI_IOCTL_EXCHANGE** and the previously set up **CSPI_XCH_PKT_T** structure as the **lpInBuffer** parameter. Also, the **nInBufferSize** parameter of `DeviceIoControl` has to be set to the size of the **CSPI_XCH_PKT_T** structure.
To wait on the operation to complete, a `WaitForSingleObject` on the assigned event handle has to be done.

Write Operation

For write opertions, a **CSPI_PACKET32** structure is used. The **data** and **address** fields have to be set accordingly. The **rw** field has to be set to **CSPI_WRITE**. A pointer to this **CSPI_PACKET32** structure must now be set as the **ptxBuf** field of the **CSPI_XCH_PKT_T** structure.
Again, the transfer is done by calling `DeviceIoControl` with the control code set to **CSPI_IOCTL_EXCHANGE** and a pointer to the **CSPI_XCH_PKT_T** structure as **lpInBuffer** parameter. After that, a call to `WaitForSingleObject` with the according event waits for completion of the operation.

Read Operation

A read operation is done analog to a write operation, but the **rw** field has to be set to **CSPI_READ**.

5.4 Serial Driver

The Serial Driver provides support for the internal UART1 of the i.MX31 (UART3 is used as debug output). The Driver is implemented as a Stream Interface Driver. It implements the PDD Layer (platform dependent) and is then linked to the Microsoft - provided serial MDD library (com_mdd2.lib) to form the Driver.

Summary

Driver Attribute	Definition
MXARM11 SOC Driver Path	`..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\SERIAL`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\SERIAL`
Import Library	`com_mdd2.lib`
Driver DLL	`csp_serial.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> Serial -> UART1 serial port`
SYSGEN Dependency	`N/A`
BSP Environment Variable	`BSP_SERIAL_UART1=1`

Software Operation

The Serial Driver implements the architecture recommended by Microsoft. For detailed information on this architecture, refer to ***Developing a Device Driver -> Windows Embedded CE Drivers -> Serial Drivers -> Serial Driver Development Concepts*** **in the Windows CE Help.**

Configuration

The UART1 Serial Driver is configured by the registry settings described below:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\COM1]
      "DeviceArrayIndex"=dword:0
      "IoBase"=dword:43F90000
      "IoLen"=dword:D4
      "Prefix"="COM"
      "Dll"="csp_serial.dll"
      "Index"=dword:1
      "Order"=dword:9

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\COM1\Unimodem]
      "Tsp"="Unimodem.dll"
      "DeviceType"=dword:0
      "FriendlyName"="MX31 COM1 UNIMODEM"
      "DevConfig"=hex: 10,00, 00,00, 05,00,00,00, 10,01,00,00,
      00,4B,00,00, 00,00, 08, 00, 00, 00,00,00,00
```

DMA Support

The Serial Driver supports the usage of DMA to minimize CPU usage. DMA for UART1 is enabled/disabled by changing '`BSP_SDMA_SUPPORT_UART1`' in the file '`PLATFORM\IMX31_PHYTEC\SRC\INC\bsp_cfg.h`'.

If DMA is enabled, the DMA buffer size can be adjusted by changing the '`SERIAL_SDMA_RX_BUFFER_SIZE`' and '`SERIAL_SDMA_TX_BUFFER_SIZE`' defines, which are also defined in '`bsp_cfg.h`'.

Serial PDD Functions

The Serial PDD Functions defined by Microsoft are mapped to Serial Driver internal functions as follows:

```
const HW_VTBL IoVTbl = {
    SerSerialInit,
    SerPostInit,
    SerDeinit,
    SerOpen,
    SerClose,
    SL_GetIntrType,
    SL_RxIntrHandler,
    SL_TxIntrHandler,
    SL_ModemIntrHandler,
    SL_LineIntrHandler,
    SL_GetRxBufferSize,
    SerPowerOff,
    SerPowerOn,
    SL_ClearDTR,
    SL_SetDTR,
    SL_ClearRTS,
    SL_SetRTS,
    SerEnableIR,
    SerDisableIR,
    SL_ClearBreak,
    SL_SetBreak,
    SL_XmitComChar,
    SL_GetStatus,
    SL_Reset,
    SL_GetModemStatus,
    SerGetCommProperties,
    SL_PurgeComm,
    SL_SetDCB,
    SL_SetCommTimeouts,
};
```

Refer to ***Developing a Device Driver -> Windows Embedded CE Drivers -> Serial Port Drivers -> Serial Port Driver Reference -> Serial Port Driver Structures -> HW_VTBL*** to determine which internal function maps to which Serial PDD function.

5.5 GPT Driver

The GPT module can be used to implement various time-related tasks with high accuracy such as external triggered timestamps and interrupts or generating periodic output. Therefore, it supports input-capture and output-compare functionality.

Summary

Driver Attribute	Definition
MXARM11 Driver Path	`..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\GPT`
Static Library	`mxarm11_gpt.lib`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\GPT`
Import Library	`N/A`
Driver DLL	`gpt.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> Timers -> GPT`
SYSGEN Dependency	`N/A`
BSP Environment Variable	`BSP_GPT=1`

Software Operation

The GPT Driver is a stream - interface driver and therefore can be accessed using the file system API (special device GPT1`:`). After opening a handle to the driver using the `CreateFile` call, communication with the device is done by issuing `DeviceIoControl` - calls. However, there are also wrapper - functions available:

HANDLE GptOpenHandle(void)
Creates a handle to the GPT stream driver.

Return value: handle to the GPT or INVALID_HANDLE_VALUE if there was an error.

BOOL GptCloseHandle(HANDLE hGpt)
Closes e GPT-handle previously created by **`GptOpenHandle.`**

HANDLE GptCreateTimerEvent(HANDLE hGpt, LPTSTR eventName)
Returns an event handle, which is triggered when a timer period has elapsed.

relevant parameters:

eventName: string containing the name of the event to be created.

BOOL GptReleaseTimerEvent(HANDLE hGpt, LPTSTR eventName)

Closes an event handle.

relevant parameters:

eventName: string containing the name of the event to be closed.

BOOL GptSetTimer(HANDLE hGpt, PGPT_TIMER_SET_PKT pGptTimerDelayPkt)
Sets the timer period for the GPT.

relevant parameters:
pGptTimerDelayPkt: pointer to a `pGptTimerDelayPkt` structure containing information for setting GPT timer delay (see structure description below)

BOOL GptStart(HANDLE hGpt)
Enables the GPT Interrupt and starts the GPT timer.

BOOL GptStop(HANDLE hGpt)
Disables the GPT Interrupt and stops the GPT timer.

Structures

```
typedef enum timerMode
{
    timerModePeriodic,
    timerModeFreeRunning
} timerMode_c;


typedef struct
{
    timerMode_c timerMode;
    UINT32 period;
} GPT_TIMER_SET_PKT, *PGPT_TIMER_SET_PKT;
```

5.6 PWM Driver

The Pulse Width Modulator (PWM) module can be used to e.g. generate tones.

Summary

Driver Attribute	Definition
MXARM11 Driver Path	..\CSP\ARM\FREESCALE\MXARM11\DRIVERS\PWM
Static Library	mxarm11_pwm.lib
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\PWM
Import Library	N/A
Driver DLL	pwm.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> PWM
SYSGEN Dependency	N/A
BSP Environment Variable	BSP_PWM=1

Software Operation

The PWM driver is a Stream Interface Driver and thus can be accessed using the filesystem APIs. Before generating signals using the PWM module, a handle to the driver must be optained using the **CreateFile** function. After that, the PWM module can accessed using the **ReadFile** and **WriteFile** functions. After using the PWM module, the handle to the driver must be closed by a call to **CloseHandle**.

Write Operations

To issue a waveform output, a call to **WriteFile** with a **PwmSample_t** structure as second parameter and the size of this structure as the third parameter is necessary. Before this, the structure must be filled accordingly (see structure description below).

Read Operations

To read out the current values of the PWM registers, it is necessary to issue a call to **ReadFile** with a pointer to an array of 32Bit Integers as second parameter and the array's size as the third parameter. After this, the array contains the values of the PWM registers in the following order:

PWM Control Register,
PWM Status Register,
PWM Interrupt Register,
PWM Sample Register,
PWM Period Register
PWM Counter Register.

Structures

```
typedef struct {
   UINT32  sample;              // 32Bit pwm sample value
   UINT32  period;              // 32Bit pwm period
   UINT32  duration;            // duration of the sample (msec)
} PwmSample_t, *pPwmSample_t;
```

5.7 NAND-Flash Driver

The NAND-Flash driver is desired to make the ST NAND512W3A2BN6 Flash available as a block device in Windows CE. It therefore implements the Flash Abstraction Layer (FAL).

Summary

Driver Attribute	Definition
MXARM11 Driver Path	N/A
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\COMMON\NANDFMD ..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\BLOCK\NANDFMD
Import Library	N/A
Driver DLL	nandfmd.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Storage Drivers -> MSFlash Drivers -> ST NAND512W3A2BN6 Nand Flash
SYSGEN Dependency	N/A
BSP Environment Variable	BSP_NAND_FMD=1

Software Operation

The NAND Flash Driver implements the Flash Media Drivers (FMD) architecture. Refer to ***Developing a Device Driver > Windows CE Drivers > Flash Media Drivers*** for more information.

5.8 One Wire Driver

The One Wire Interface allows the communication with devices using only one (actually two) electrical lines, for example the Dallas DS1921 temperature - sensor.

Summary

Driver Attribute	Definition
MXARM11 Driver Path	`..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\OWIRE`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\OWIRE`
Import Library	`N/A`
Driver DLL	`owire.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> One-wire Interface -> One-wire`
SYSGEN Dependency	`N/A`
BSP Environment Variable	`BSP_OWIRE=1`

Software Operation

The One Wire Driver is a stream - interface driver and therefore can be accessed using the file system API (special device `WIR1:`). After opening a handle to the driver using the `CreateFile` call, communication with the device is done by issuing `DeviceIoControl` - calls. However, there are also wrapper - functions available:

`HANDLE OwireOpenHandle()`
Opens a handle to the onewire-module, which can be used in further calls to the device.

`BOOL OwireCloseHandle(HANDLE hOwire)`
After using the device, it's corresponding handle can be closed with this function.

`BOOL OwireReset(HANDLE hOwire)`
Performs a software reset on the owire-module.

`BOOL OwireResetPresencePulse(HANDLE hOwire)`
Before first using an owire-device, it has to be resetted and it's presence has to be detected. This is accomplished by calling the `OwireResetPresencePulse` function. If this call returns `TRUE`, the device is ready to receive commands.

BOOL OwireRead(HANDLE hOwire, BYTE *readBuf, DWORD bytesToRead)

Use this function to receive responses or data from a device.

BOOL OwireWrite(HANDLE hOwire, BYTE *writeBuf, DWORD bytesToWrite)

This function is used to write commands or data to an one-wire device. To do so, fill a buffer with the command(s) or data which is to be send and set the `bytesToWrite` - parameter to the size of the buffer. If the write operation succeeds, the call will return `TRUE`.

5.9 Audio Driver

he audio driver is used to playback or record audio using the MC13783 PMIC.

Summary

Driver Attribute	Definition
MXARM11 Driver Path	`..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\WAVEDEV` `..\PLATFORM\COMMON\SRC\SOC\FREESCALE\PMIC\MC13783\SDK`
SOC static library	`mxarm11_wavedev.lib.lib` `pmicSdkCsp_mc13783.lib`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\PMIC\MC13783\PDK` `..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\WAVEDEV\MC13783`
Import Library	`N/A`
Driver DLL	`wavedev_MC13783.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> Audio -> MC13783 Audio Driver` `CoreOS -> Windows CE devices -> Graphics and Multimedia Technologies -> Audio -> Waveform Audio`
SYSGEN Dependency	`N/A`
BSP Environment Variable	`BSP_AUDIO_MC13783=1` `BSP_PMIC_MC13783=1`

Software Operation

The audio driver conforms to the architecture for audio drivers recommended by Microsoft. For a detailed description, read ***Developing a Device Driver -> Windows CE Drivers -> Audio Drivers -> Audio Driver Development Concepts*** in the WindowsCE help.

5.10 MMC/SD Driver

The Secure Digital Host Controller (SDHC) supports Multimedia Cards (MMC), Secure Digital Cards (SD) and Secure Digital I/O Cards (SDIO).
The SDHC driver provides the interface between Microsoft's SD Bus Driver and the i.MX31 SDHC.

Summary

Driver Attribute	Definition
MXARM11 Driver Path	`..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\SDHC`
SOC static library	`mxarm11_sdhc.lib`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\SDHC`
Import Library	`N/A`
Driver DLL	`sdhc.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> SD Host Controller -> SD Host Controller 1`

SYSGEN Dependency	`SYSGEN_SD_MEMORY=1`
	`SYSGEN_BTH=1`
	`SYSGEN_BTH_SDIO_ONLY=1`
	`BSP_NIC_WLAN6060_SDIO=1`

BSP Environment Variable	`BSP_SDHC1=1`

Software Operation

The SDHC Driver follows the architecture for Secure Digital Host Controller drivers recommended by Microsoft. For more information, look at ***Developing a Device Driver -> Windows Embedded CE Drivers -> Secure Digital Card Drivers -> Secure Digital Card Driver Development Concepts*** in the Windows CE Help.

Required Catalog Items

for SD and MMC Memory Card Support:

Catalog −> Device Drivers −> SDIO −> SD Memory

for Bluetooth SDIO Support:

Catalog −> Core OS −> Communication Services and Networking −> Networking −> Personal Area Network −>Bluetooth −> Bluetooth Protocol Stack with Transport Driver Support −> Bluetooth Stack with Universal Loadable Driver

Catalog −> Core OS −> Communication Services and Networking −> Networking −> Personal Area Network > Bluetooth −> Bluetooth Protocol Stack with Transport Driver Support −> Bluetooth Stack with Integrated Driver

for Wi-Fi SDIO Support:

Catalog −> Device Drivers −> SDIO −> SDIO Clients −> SDIO WiFi

Registry Settings

```
#if (defined BSP_SDHC1)
[HKEY_LOCAL_MACHINE\Drivers\SDCARD\ClientDrivers\Class\SDMemory_Class]
   "BlockTransferSize"=dword:100  ; Overwrite from default 64 blocks.
;   "SingleBlockWrites"=dword:1  ; alternatively force the driver to use single
block access

[HKEY_LOCAL_MACHINE\Drivers\SDCARD\ClientDrivers\Class\MMC_Class]
  "BlockTransferSize"=dword:100  ; Overwrite from default 64 blocks.
;  "SingleBlockWrites"=dword:1  ; alternatively force the driver to use single
block access

[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\MMC]
    "Name"="MMC Card"
    "Folder"="MMC"

[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\SDMemory]
    "Name"="SD Memory Card"
    "Folder"="SD Memory"
#endif

IF BSP_SDHC1
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\SDHC_ARM11_1]
    "Order"=dword:21
    "Dll"="sdhc.dll"
    "Prefix"="SDH"
    "ControllerISTPriority"=dword:64
    "Index"=dword:1
ENDIF ;BSP_SDHC1
```

5.11 PCMCIA Driver

The PCMCIA driver implements a socket driver for the WindowsCE PCMCIA stack. This dirver supports 16Bit PCMCIA cards and 32Bit Cardbus cards.

Summary

Driver Attribute	Definition
SOC Driver Path	`..\PLATFORM\COMMON\SRC\ARM\FREESCALE\MX31\DRIVERS\PCCARD`
SOC static library	`mx31_pcc_lib.lib`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\PCCARD`
Import Library	`pcc_com.lib, cspddk_lib, coredll.lib, ceddk.lib`
Driver DLL	`pcc_mx31.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> PC Card Host -> i.MX31 PCMCIA (PC Card)`
SYSGEN Dependency	`SYSGEN_ATADISK=1` `SYSGEN_TFAT=0`
BSP Environment Variable	`BSP_PCMCIA=1`

Software Operation

The PCMCIA socket driver follows the architecture for PC Card Drivers recommended by Microsoft. Details can be found under ***Developing a Device Driver −> Windows CE Drivers −> PC Card Drivers −> PC Card Development Concepts.***

The socket driver exports the following functions:

- Init()
- Deinit()
- PowerUp
- PowerDown

The socket service API's are encapsulated in a SS_SOCKET_SERVICE structure. It contains the function pointers to the following functions, which allow communication between the card service driver (Pcc_Serv.dll) und the socket driver:

- PDCardInquireSocket
- PDCardGetSocket
- PDCardSetSocket
- PDCardResetSocket
- PDCardInquireWindow
- PDCardGetWindow
- PDCardSetWindow
- PDCardAccessMemory
- PDGetPowerEntry

For a detailed description of this functions, refer to the PC Card Reference in the WindowsCE Help.

5.12 USB Host Driver

The USB Host Driver provides USB 2.0 host support using the USB High Speed Host (H2) of the i.MX31.
The driver has class support for mass storage, HID, printer and RNDIS clients.

Summary

Driver Attribute	Definition
MXARM11 Driver Path	..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\USBH
SOC static library	mx31_ehci_lib.lib mx31_Ehcdmdd.lib mx31_hcd2lib.lib
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\USBH\USBH2
Import Library	N/A
Driver DLL	hcd_hsh2.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> USB Devices -> USB Host Devices -> USB High Speed Host 2
SYSGEN Dependency	SYSGEN_USB=1
BSP Environment Variable	BSP_USB=1 BSP_USB_HSH2=1

The host driver requires one or more class drivers to be loaded (see below).

Required Catalog Items

For Human Interface Devices (HID):

Core OS−>Windows CE devices −>Core OS Services −> USB Host Support −> USB Human Input Device (HID) Class Driver

For Printers:

Core OS−>Windows CE devices −>Core OS Services −> USB Host Support −> USB Printer Class Driver

For RNDIS:

Core OS−>Windows CE devices −>Core OS Services −> USB Host Support −> USB Remote NDIS Class Driver

For Mass Storage:

Core OS−>Windows CE devices −>Core OS Services −> USB Host Support −> USB (mass) Storage Class Driver

Note that sometimes additional Catalog Items have to be included in order to get the desired functionality (i.e. filesystem drivers, printer protocols, keyboard layouts etc...)

Software Operation

The driver implements the Windows USB Software Architecture. For more information, refer to ***Developing a Device Driver −> Windows CE Drivers −> USB Host Drivers*** and ***Developing a Device Driver −> Windows CE Drivers −> USB Host Drivers −> USB Host Controller Drivers −> USB Host Controller Driver Development Concepts*** in the Windows CE help.

5.13 USB OTG Driver

The OTG driver provides USB host and device support for the i.MX31 USB "On The Go" (OTG) port. It will automatically select host or device functionality depending on the USB cable configuration. The driver consists of three parts: the USB OTG host controller driver, the USB client driver and the USB transceiver controller driver (for host/client switching).

OTG Client Summary

Driver Attribute	Definition
MXARM11 Driver Path	..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\USBD
SOC static library	mxarm11_usbfn.lib
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\USBD
Import Library	UFNMDDBASE.lib
Driver DLL	usbfn.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> USB Devices -> USB High Speed OTG Device -> USB High Speed OTG Port Full OTG Function
SYSGEN Dependency	SYSGEN_USBFN=1
BSP Environment Variable	BSP_USB=1 BSP_USB_HSOTG_CLIENT

When in client mode, there must be configured a function driver to use (mass storage, serial and rndis function drivers are supported by Windows CE). In order to use ActiveSync, the serial function driver must be selected.

OTG Host Summary

Driver Attribute	Definition
MXARM11 Driver Path	..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\USBXVR ..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\USBH\EHCI
SOC static library	N/A
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\USBH\HSOTG
Import Library	mx31_ehci_lib.lib mx31_Ehcdmdd.lib mx31_hcd2lib.lib
Driver DLL	hcd_hsotg.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> USB High Speed OTG Device -> USB High Speed OTG Port Full OTG Function or Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> USB High Speed OTG Device -> USB High Speed OTG Pure Host Function

SYSGEN Dependency	SYSGEN_USBFN=1
BSP Environment Variable	BSP_USB=1 BSP_USB_HSOTG_HOST

The OTG host driver requires one or more class drivers to be loaded, see section 5.13.

OTG Transceiver Summary

Driver Attribute	Definition
MXARM11 Driver Path	..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\USBXVR
SOC static library	mx31_xvc.lib
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\USBXVR
Import Library	N/A
Driver DLL	imx_xvc.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> USB High Speed OTG Device -> USB High Speed OTG Port Full OTG Function
SYSGEN Dependency	SYSGEN_USBFN=1
BSP Environment Variable	BSP_USB=1 BSP_USB_HSOTG_CLIENT BSP_USB_HSOTG_HOST BSP_USB_HSOTG_XVC

Software Operation

USB OTG Host Controller Driver

The USB OTG host controller driver is part of the Windows USB software standard. See section 5.13 for help on getting more information.

USB OTG Client Driver

The USB OTG client driver is part of the Windows USB software standard. For more information, refer to ***Developing a Device Driver −> Windows CE Drivers −> USB Function Drivers −> USB Function Controller Drivers*** and ***Developing a Device Driver −> Windows CE Drivers −> USB Function Client Drivers*** in the Windows CE help.

5.14 I²C Driver

The I²C module can be used to communicate with one or more i²c devices, such as EEPROM, RTC or Camera devices.

Summary

Driver Attribute	Definition
MXARM11 Driver Path	`..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MXARM11\DRIVERS\I2C`
SOC Driver Path	`..\PLATFORM\COMMON\SRC\SOC\FREESCALE\MX31\DRIVERS\I2C`
Platform Driver Path	`..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\I2C`
SOC static Libraries	`mxarm11_i2c.lib, mx31_i2c.lib`
Import Library	`N/A`
Driver DLL	`i2c.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> I2C Bus`
SYSGEN Dependency	`N/A`
BSP Environment Variable	`BSP_I2CBus=1`

Software Operation

The I²C driver is a stream-interface driver and thus can be accessed through the filesystem APIs. Therefore, a handle to the driver must first be created using the `CreateFile` - function. After that, commands and data can be passed to or from the driver by using the `DeviceIoControl` function with specific IOCTL codes (explained below). After using the I²C bus, the device handle has to be cosed using the `CloseHandle` function.

IOCTL codes

`I2C_IOCTL_SET_SLAVE_MODE`

Requires no parameters, sets the I²C device to slave mode.

`I2C_IOCTL_SET_MASTER_MODE`

Requires no parameters, sets the I²C device to slave mode.

I2C_IOCTL_IS_MASTER

Determines, if the I^2C device is currently in master mode.

relevant Parameters:

lpOutBuffer
Pointer to a BYTE which will hold the return value: TRUE if device is in master mode, otherwise FALSE

nOutBufferSize
Size in Bytes of the return value (should be 1 Byte).

I2C_IOCTL_IS_SLAVE

Same as **I2C_IOCTL_IS_MASTER,** but instead checking if the device is in slave mode.

I2C_IOCTL_GET_CLOCK_RATE

Retrieves the clock rate devisor. The actual clock rate is platform dependent.

relevant Parameters:

lpOutBuffer
Pointer to the divisor index. Refer to I^2C specification for more information.

nOutBufferSize
Size in Bytes of the divisor index.

I2C_IOCTL_SET_CLOCK_RATE

Initializes the I^2C device with the given clockrate. Note that this is not the peripheral frequency, but the clock rate devisor index.

relevant Parameters:

lpInBuffer
Pointer to the divisor index. Refer to I^2C specification for more information.

nInBufferSize
Size in Bytes of the divisor index.

I2C_IOCTL_SET_FREQUENCY

Estimates the nearest clock rate acceptable for the device and initializes the device to use the estimated clock rate devisor.

relevant Parameters:

lpInBuffer

Pointer to the desired I^2C frequency.

nInBufferSize

Size in Bytes frequency requested.

I2C_IOCTL_SET_SELF_ADDR

Initializes the device with the given address.

relevant Parameters:

lpInBuffer

Pointer to the expected I^2C device address (0x00 to 0x7F).

nInBufferSize

Size in Bytes of the I^2C device address.

I2C_IOCTL_GET_SELF_ADDR

Retrieves the address of the I^2C device (only meaningfull if device is in slave mode).

relevant Parameters:

lpOutBuffer

Pointer to the current I^2C device address (0x00 to 0x7F).

nOutBufferSize

Size in Bytes of the I^2C device address.

I2C_IOCTL_TRANSFER

Performs a sequential transfer (read or write) of one or more data-packets to or from a device. Expects a I2C_TRANSFER_BLOCK structure containing an array of I2C_PACKET - structures to be transferred.
This array is performed sequentially. An I2C START command is issued before the first packet transmission. Further, if the transfer direction or slave address changes between different packages of the array, an I2C REPEATED START command is automatically inserted. After transmission of the last packet in the array, an I2C STOP command is issued.

relevant Parameters:

lpInBuffer

Pointer to an I2C_TRANSFER_BLOCK structure (explained below).

nInBufferSize

Size in Bytes of the I2C_TRANSFER_BLOCK..

I2C_IOCTL_RESET

Performs a hardware reset.

Structures

I2C_TRANSFER_BLOCK

```
typedef struct
{
    I2C_PACKET *pI2CPackets;        //array containing the packets to be
                                    //transferred
    INT32 iNumPackets;              //number of packet structures in the array
} I2C_TRANSFER_BLOCK, *PI2C_TRANSFER_BLOCK;
```

I2C_PACKET

```
typedef struct
{
    BYTE byAddr;        // I2C slave device address for this I2C operation
    BYTE byRW;          // Read = I2C_READ or Write = I2C_WRITE
    PBYTE pbyBuf;       // Message Buffer
    WORD wLen;          // Message Buffer Length
    LPINT lpiResult;    // Contains the result of last operation
} I2C_PACKET, *PI2C_PACKET;
```

5.15 Ethernet Driver

The SMSC9217 Fast Ethernet Driver provides the ability to use a wide range of networking services on the device.

Summary

Driver Attribute	Definition
MXARM11 SOC Driver Path	N/A
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\SMSC9118
IC-specific SOC Driver Path	N/A
Import Library	ndis.lib
Driver DLL	smsc9118.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> Networking -> Local Area Networking (LAN) devices -> SMSC 9118 - NDIS
SYSGEN Dependency	SYSGEN_NDIS=1 SYSGEN_TCPIP=1 SYSGEN_WINSOC=1
BSP Environment Variable	BSP_ETHER_SMSC9118

Software Operation

The SMSC9217 Ethernet driver confirms to the NDIS 4.0 specification by Microsoft for the miniport network drivers.

5.16 CAN Controller Driver

The SJA1000 CAN – Driver provides the ability to connect the Evaluation Board to a CAN-Bus with various Bitrates and standard- or extended frame formats.

Summary

Driver Attribute	Definition
MXARM11 SOC Driver Path	`N/A`
Platform Driver Path	`N/A`
IC-specific SOC Driver Path	`N/A`
Import Library	`N/A`
Driver DLL	`SJA1000.dll` `gfcan32ce.dll`
Required Catalog Items	`Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> CAN -> Driver for SJA1000 CAN-Controller`
BSP Environment Variable	`BSP_SJA1000`

Software Operation

Please contact Phytec for information regarding the CAN driver.

5.17 PMIC Driver

The MC13783 companion is used for a variety of functionality, for instance power management, touchscreen controller, audio, battery charging and A-D conversion.

Summary

Driver Attribute	Definition
SOC Driver Path	..\PLATFORM\COMMON\SRC\ARM\FREESCALE\PMIC\MC13783\PDK
SOC static library	pmicPdkCsp_MC13783.lib
Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\PMIC\MC13783\PDK
Import Library	N/A
Driver DLL	pmicPdkCsp_MC13783.dll
Required Catalog Items	N/A
SYSGEN Dependency	N/A
BSP Environment Variable	BSP_PMIC_MC13783=1

Software Operation

For low-level access to the MC13783 PMIC there exists a stream interface driver. This driver exposes the functionality of reading/writing MC13783 registers, enabling/disabling interrupts and handling ADC conversions through some custom IOControlCodes.
To issue this IOControls, there must first be obtained a handle to the PMIC-driver by calling **`CreateFile`** with the special filename **`PMI1:`**.
After that, it is possible to use the following IOControlCodes for **`DeviceIOControl():`**

IOControls

PMIC_IOCTL_LLA_READ_REG
reads a PMIC register.

relevant parameters:
lpInBuffer: index of the register

lpOutBuffer: Long Pointer to a buffer that receives the output data

PMIC_IOCTL_LLA_WRITE_REG
wites to a PMIC register.

relevant parameters:
lpInBuffer: index of the register

lpOutBuffer: Pointer to a buffer that contains the data which is to be written

PMIC_IOCTL_LLA_INT_REGISTER
registers a PMIC interrupt.

relevant parameters:
lpInBuffer: PMIC_PARAM_INT_REGISTER struct containing interrupt id and event

PMIC_IOCTL_LLA_INT_DEREGISTER
de-registers a PMIC interrupt.

relevant parameters:
lpInBuffer: PMIC_PARAM_INT_REGISTER struct containing interrupt id and event

PMIC_IOCTL_LLA_INT_COMPLETE
completes (re-enables) a PMIC interrupt.

relevant parameters:
lpInBuffer: PMIC_INT_ID containing interrupt id to complete

PMIC_IOCTL_LLA_INT_ENABLE
enables a PMIC interrupt.

relevant parameters:
lpInBuffer: PMIC_INT_ID containing interrupt id to enable

PMIC_IOCTL_LLA_INT_DISABLE
enables a PMIC interrupt.

relevant parameters:
lpInBuffer: PMIC_INT_ID containing interrupt id to disable

5.18 EEPROM Driver

The EEPROM – Driver can be used to store small amounts of data (up to 4096 Byte) in a persistant way using the 24WC32 EEPROM on the Module.

Summary

Driver Attribute	Definition
MXARM11 SOC Driver Path	N/A

Platform Driver Path	..\PLATFORM\IMX31_PHYTEC\SRC\DRIVERS\24WC32
IC-specific SOC Driver Path	N/A
Import Library	N/A
Driver DLL	24wc32.dll
Required Catalog Items	Third Party -> BSPs -> iMX31_Phytec -> Device Drivers -> EEPROM -> Catalyst 24WC32 EEPROM Driver
BSP Environment Variable	BSP_EEPROM_24WC32 BSP_I2CBUS

Software Operation

The EEPROM driver implements the Stream Interface. To use the driver, a handle to the device must be created using the **CreateFile** function with the device name 'EEP1'. All other commands to the device can be issued using the **DeviceIoControl** function with the IOCTLs mentioned below:

IOCTL_READ

Description:

reads a specified amount of data from the EEPROM to a buffer.

Parameters:

pBufIn [IN] pointer to an EEPROM_TRANSFER structure containing the EEPROM internal address, the length of the buffer and a pointer to the buffer (see below)

pBufOut [OUT] pointer to an unsigned int, will be set to the amount of Bytes which were actually read

Return Value:

BOOL TRUE if read operation succeeded, else FALSE

IOCTL_WRITE

Description:

writes a specified amount of data from a buffer to EEPROM.

Parameters:

`pBufIn` [IN] pointer to an `EEPROM_TRANSFER` structure containing the EEPROM internal address, the length of the buffer and a pointer to the buffer (see below)

Return Value:

`BOOL` `TRUE` if write operation succeeded, else `FALSE`

Structures

The EEPROM driver defines the following structure to describe a read or write operation:

```
typedef struct
{
     UINT32         Addr;          //EEPROM internal address
     UINT8*         pBuffer;       //pointer to data buffer
     UINT32         Length;        //amount of Bytes to read/write
}EEPROM_TRANSFER;
```

6. Configuring the Splashscreen

The Bootloader (EBOOT) supports displaying a custom splashscreen - image on startup. Currently this feature is only implemented for the Sharp LQ035Q7DH06 panel. The bitmap which is to be displayed must be present in the following format:

- Resolution: 240x320 Pixel
- Format: 16-Bit Bitmap (565)

An example bitmap can be found in the 'Splashscreen' folder.

This bitmap has to be programmed to flash address `0xA0060000`. This is done in the same way as programming the bootloader to the device, although using address `0xA0060000` instead of `0xA0000000` (EBOOT location) and choosing the bitmap file instead of the EBOOT - image.

If the bootloader can not find a bitmap in the correct format or no image data at all, a white screen is displayed.

CAUTION: If the bitmap is not in the correct format or it's size is greater than 512KB, unpredictable behavior can occur! Please use only bitmaps in the correct format, as explained above!