NetDCU Device Drivers

Windows Embedded CE 6.0

Version 1.06 Date: 2010-08-24

F&S Elektronik Systeme GmbH
Untere Waldplätze 23
D-70569 Stuttgart
Fon: +49(0)711-123722-0
Fax: +49(0)711-123722-99

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2009-02-16	1.00		A	*	First version	DK
2009-03-25	1.01	9	A	2	Analogue Input Configuration Table	CZ
2009-04-07	1.02	10	M	4	PWM register settings	CZ
2009-04-15	1.03	9,10	M	3	Interrupt configuration table corrected	DK
2009-06-09	1.04	*	M	3	Added example to access individual pins	DK
2010-02-05	1.05	10	M	12	Touchpanel register settings	CZ
2010-08-24	1.06	9,10	A,M	12	Touchpanel register settings	CZ

V	Version
A,M,R	Added, Modified, Removed
Au	Author

Contents

1 Windows CE Stream Interface Driver

All NetDCU device drivers are implemented as Windows CE Stream Interface Driver. Thus you can access these drivers via the File System and the respective File API (CreateFile, WriteFile, ReadFile, SetFilePointer, DeviceIoControl).

A stream interface driver receives commands from the Device Manager and from applications by means of file system calls. The driver encapsulates all of the information that is necessary to translate those commands into appropriate actions on the devices that it controls. All stream interface drivers, whether they manage built-in devices or installable devices, or whether they are loaded at boot time or loaded dynamically, have similar interactions with other system components. The following illustrations show the interactions between system components for a generic stream interface driver that manages a built-in device.

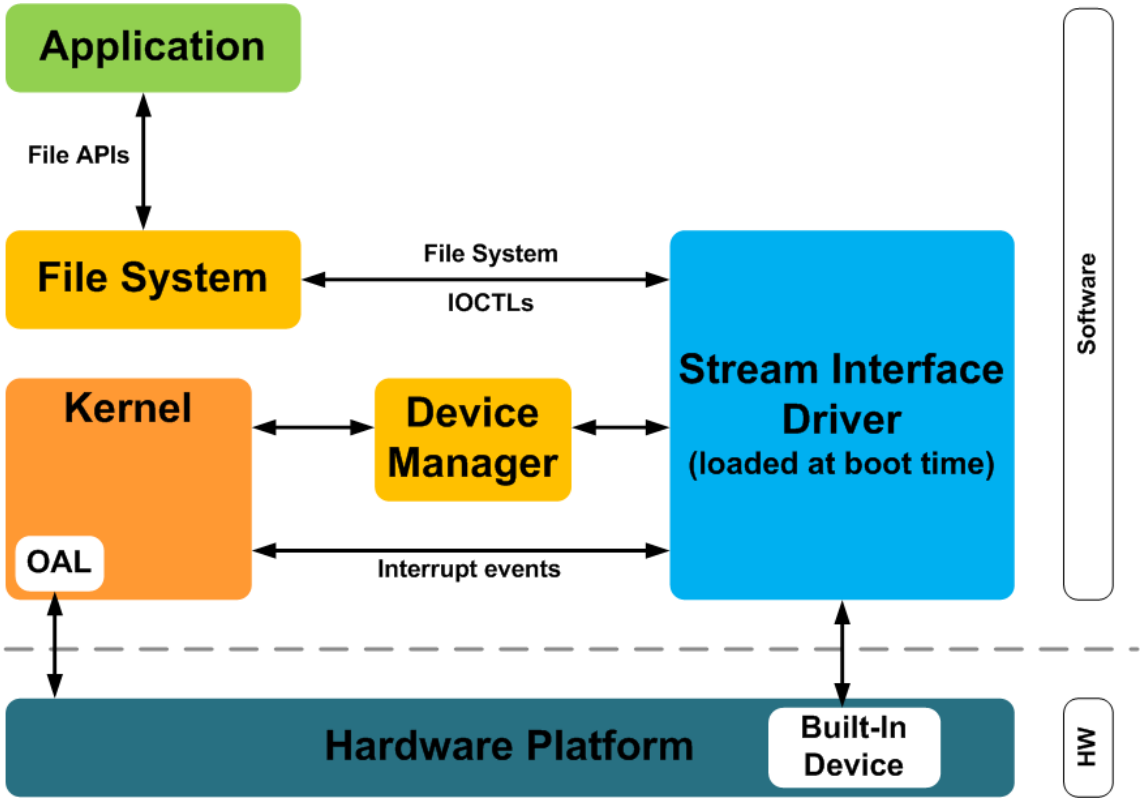


Figure 1: Windows CE: Stream Interface Driver Architecture

2 Analogue Input

Implemented on NetDCU: 9,10

NetDCU has 4 analogue inputs. They are internally connected with a resistor (47K) to ground. This four inputs can be read with this driver. You must install one copy of the driver for each input. The selection of the channel can be done with the registry key *Channel*.

Installation of the driver is done by setting some registry values under the following registry key:

```
[HKLM\Drivers\BuiltIn\ANALOGIN]
```

Required settings:

Key	Value	Comment
"Prefix"	“AIN“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“ANALOGIN.DLL“	name of the DLL with the driver
“Order“	Dword:0x97	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
“Index“	Dword:1	This value specifies the device index, a value from 0 through 9.
“Flags”	Dword:0	4: Disabeld from loading
“Ioctl“	Dword:4	Call post-initialisation function.
“Channel“	Dword:n	Number of the analogue channel. See Table Channel.
“FriendlyName“	“Analogue input driver for NetDCU“	
“Debug”	Dword:0|4	Set to 4 to get list of registry settings at serial debug port. Default: 0

Table 1: Analogue Input: Registry

Table Channel NetDCU10:

Channel	Description
0x00	Reads value from analogue input 0 (Connector J7, AD0)
0x01	Reads value from analogue input 1 (Connector J7, AD1)
0x02	Reads value from analogue input 2 (Connector J7, AD2)
0x03	Reads value from analogue input 3 (Connector J7, AD3)

Table 2: Analogue Input: Channels

Table Channel NetDCU9:

Channel	Description
0x10	Reads value from analogue input 0 (Connector J7, AD0)
0x14	Reads value from analogue input 1 (Connector J7, AD1)
0x18	Reads value from analogue input 2 (Connector J7, AD2)
0x1C	Reads value from analogue input 3 (Connector J7, AD3)

Table 3: Analogue Input: Channels

Programming Example:

A. Open one analogue channel:

```
HANDLE hAIN;
hAIN = CreateFile( _T("AIN1:"),GENERIC_READ, 0, NULL, OPEN_EXISTING
                               ,FILE_ATTRIBUTE_NORMAL, NULL );
if( hAIN == INVALID_HANDLE_VALUE )
{
  ERRORMSG(1,L"Can not open AIN1. LastError = 0x%x\r\n",GetLastError()));
  return(FALSE);
}
```

Listing 1: Analogue Input: Open channel

B. Read data from previously opened channel:

```
unsigned short data;
DWORD dwSamples = 1;
ReadFile( hAIN, data, dwSamples, &dwSamples, NULL );
if( dwSamples != 1 )
{
  ERRORMSG(1,L"Can not read from AIN1. LE = 0x%x\r\n",GetLastError()));
}
```

Listing 2: Analogue Input: reading samples

C. Select another channel without changing registry:

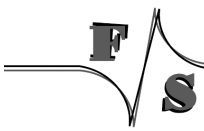
```
int nChannel = 0x0;
SetFilePointer( hAIN, nChannel, 0, FILE_BEGIN );
```

Listing 3: Analogue Input: changing channel from application

D. Closing the analogue channel:

```
CloseHandle(hAIN);
```

Listing 4: Analogue Input: closing a channel



3 Digital I/O

Implemented on NetDCU: 9,10

NetDCU has programmable I/O lines at connector J5. You have to use these driver to configure and access the I/O lines.

Installation of the driver is done by setting some registry values under the following registry key:

```
[HKLM\Drivers\BuiltIn\DIGITALIO]
```

Required settings:

Key	Value	Comment
"Prefix"	“DIO“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“DIGIO.DLL“	Name of the DLL with the driver
“Order“	Dword:0x97	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
“Index“	Dword:1	This value specifies the device index, a value from 0 through 9.
“Ioctl“	Dword:4	Call post-initialisation function.
“Port“	Dword:n	0,1 or 2
UseAsIO	Dword:n	1 = The corresponding pin is used as general purpose I/O. One bit for each I/O pin.
DataDir	Dword:n	Data Direction. 0 = The corresponding pin is an input. 1 = The corresponding pin is an output. One bit for each I/O pin.
DataInit	Dword:n	Default value of the output pin after driver initialization.
IRQCfg0	Dword:n	Interrupt configuration register 0.
IRQCfg1	Dword:n	Interrupt configuration register 1.
IRQCfg2	Dword:n	Interrupt configuration register 2.
“FriendlyName“	Digital I/O driver for NetDCU“	
“Debug”	Dword:0|4	Set to 4 to get list of registry settings at serial debug port. Default: 0

Table 4: Digital I/O: Registry

Programming Example:

Headerfile:

```
#include <dio_sdk.h>
```

Listing 5: Digital I/O: Headerfile

A. Opening a digital port

```
HANDLE hDIO = CreateFile(L"DIO1:",GENERIC_READ|GENERIC_WRITE, 0, NULL, OPEN_EXISTING
                                  ,FILE_ATTRIBUTE_NORMAL,NULL );

if(INVALID_HANDLE_VALUE == hDIO)
{
  ERRORMSG(1,(L"INVALID HANDLE VALUE\r\n"));
  return -1;
}
```

Listing 6: Digital I/O: Open port

B. Write data to port

```
unsigned char data = 0xAA;
DWORD dwBytesWrite = 1;
WriteFile( hDIO, &data, dwBytesWrite, &dwBytesWrite, NULL );
if( dwBytesWrite != 1 )
{
  ERRORMSG(1,L"Can not write to DIO1. LE = 0x%x\r\n",GetLastError()));
}
```

Listing 7: Digital I/O: Write port

C. Change port

```
LONG lDistance = 1;
SetFilePointer( hDIO, lDistance, NULL, FILE_BEGIN);
```

Listing 8: Digital I/O: Change port

D. Get / Set / Clear individual pin

```
DWORD dwOutCount = 0;
DWORD dwPin = 7;
BYTE byPinLevel = 0xAA;
/*
 *      Get level of pin.
 *      dwPin = pin of interest (7 for GPIO7 which is Pin#2 on J5) = input parameter.
 *      byPinLevel = level of pin = output parameter. 0 = 0V, 1 = 3.3V
 */
DeviceIoControl(g_hDio, IOCTL_DIO_GET_PIN, &dwPin, sizeof(BYTE), &byPinLevel, sizeof(BYTE),
                &dwOutCount, NULL);
DeviceIoControl(g_hDio, IOCTL_DIO_SET_PIN, &dwPin, sizeof(BYTE), NULL, 0, &dwOutCount, NULL);
DeviceIoControl(g_hDio, IOCTL_DIO_CLR_PIN, &dwPin, sizeof(BYTE), NULL, 0, &dwOutCount, NULL);
```

Listing 9: Digital I/O: Access individual pin

E. Request interrupt on J5 pin #2

```
WAITIRQ cWait;                          //defined in dio_sdk.h
cWait.bType = FALSE;                    //set always to false.
cWait.dwTimeOut = 0;                    //timeout to be blocked (_WAIT_IRQ) by DeviceIoControl.
cWait.dwIOPin = 7;                      //pin #2 on J5 is GPIO7.

DWORD dwRet=-1;
```

```
if(!DeviceIoControl(hDIO, IOCTL_DIO_REQUEST_IRQ, &cWait.dwIOPin, sizeof(DWORD), NULL, 0
                                                , &dwRet, NULL))
   ERRORMSG(1,(L"IOCTL_DIO_REQUEST_IRQ\r\n"));
```

Listing 10: Digital I/O: Request interrupt

F. Wait for interrupt

```
cWait.dwTimeOut = 10000;              //timeout (msec) to be blocked by
                                      //DeviceIoControl. INFINITE is possible.
DWORD dwResult=-1;                    //Result: WAIT_ABANDONED, WAIT_OBJECT_0, WAIT_TIMEOUT.
                                     //For more information see WaitForSingleObject API.
if(!DeviceIoControl(hDIO,IOCTL_DIO_WAIT_IRQ, &cWait, sizeof(WAITIRQ), &dwResult, sizeof(DWORD)
                                    , &dwRet, NULL))
   ERRORMSG(1,(L"IOCTL_DIO_WAIT_IRQ\r\n"));
```

Listing 11: Digital I/O: Wait for interrupt

G. Signal that interrupt processing has been completed

```
if(!DeviceIoControl(hDIO,IOCTL_DIO_INTDONE_IRQ, &cWait.dwIOPin, sizeof(DWORD), NULL, 0
                                              , &dwRet, NULL))
   ERRORMSG(1,(L"IOCTL_DIO_INTDONE_IRQ\r\n"));
```

Listing 12: Digital I/O: InterruptDone

H. Release interrupt

```
if(!DeviceIoControl(hDIO,IOCTL_DIO_RELEASE_IRQ, &cWait.dwIOPin, sizeof(DWORD), NULL, 0
                                              , &dwRet, NULL))
   ERRORMSG(1,(L"IOCTL_DIO_RELEASE_IRQ\r\n"));
```

Listing 13: Digital I/O: Release interrupt

I. Closing port

```
CloseHandle(hDIO);
```

Listing 14: Digital I/O: Closing port

3.1 Interrupt configuration

IRQCfg2	IRQCfg1	IRQCfg0	Function	Note
0	0	0	Interrupt Disabled	
0	0	1	Rising Edge Enabled	
0	1	0	Falling Edge Enabled	
0	1	1	Rising and Falling Edge Enabled	
1	0	0	Interrupts Disabled	
1	0	1	High Level Enabled	Only on NetDCU10
1	1	0	Low Level Enabled	Only on NetDCU10

Table 5: Digital I/O: Interrupt configuration

3.2 Port description

Port 0:

Bit	7	6	5	4	3	2	1	0
Pin	2	3	4	5	6	7	8	9
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
UseAsIO Bit	7	6	5	4	3	2	1	0
DataDir Bit	7	6	5	4	3	2	1	0
DataInit Bit	7	6	5	4	3	2	1	0
IRQCfg0..2	7	6	5	4	3	2	1	0

Table 6: Digital I/O: Port 0

Port 1:

Bit	7	6	5	4	3	2	1	0
Pin	---	---	---	---	10	11	13	15
R/W	---	---	---	---	R/W	R/W	R/W	R/W
UseAsIO Bit	15	14	13	12	11	10	9	8
DataDir Bit	---	---	---	---	11	10	9	8
DataInit Bit	---	---	---	---	11	10	9	8
IRQCfg0..2	---	---	---	---	---	---	---	---

Table 7: Digital I/O: Port 1

Port 2:

Bit	7	6	5	4	3	2	1	0
Pin	17	18	19	20	21	22	23	24
R/W	R	R	R	R	R	R	R	R
UseAsIO Bit	23	22	21	20	19	18	17	16
DataDir Bit	---	---	---	---	---	---	---	---
DataInit Bit	---	---	---	---	---	---	---	---
IRQCfg0..2	---	---	---	---	---	---	---	---

Table 8: Digital I/O: Port 2

4 PWM output

Implemented on NetDCU: 10

NetDCU10 has 2 PWM outputs. One is controlled by the display driver (contrast voltage) and one can be controlled by the PWM driver.

Installation of the driver is done by setting some registry values under the following registry key:

```
[HKLM\Drivers\BuiltIn\PWM]
```

Required settings:

Key	Value	Comment
"Prefix"	“PWM“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“ND10_PWM.DLL“	Name of the DLL with the driver
“Order“	Dword:0x97	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
“Index“	Dword:1	This value specifies the device index, a value from 0 through 9.
“Mode”	Dword:0|1	0: Absolute mode. Values range between 0 and “Steps” 1: Percent mode Values between 0 and 100%. Default: 1
“Steps”	Dword:0..0xFFFF	Amount of clocks in one frame. Default: 0xFFF
“Freq”	Dword:	Clock frequency Default: 300000Hz
“FriendlyName“	“PWM driver for NetDCU“	
“Debug”	Dword:0|4	Set to 4 to get list of registry settings at serial debug port. Default: 0

Table 9: PWM: Registy

Note:

After opening the channel you can call WriteFile() to set the high phase. Use ReadFile() to read back the current value. The type of pointer is BYTE for Mode 1 and WORD for Mode 0.
Please take a look at file pwm_sdk.h for additional IOCTL’s.

5 LC Display driver

Implemented on NetDCU: 9,10

NetDCU has a very flexible and powerful interface for LCD and EL displays. The driver is fully configurable over the Window CE registry. The user has the possibility to adjust the driver to a new display by himself.

Registry Key:

NetDCU	Key
9	[HKLM\Drivers\Display\SMIVGX]
10	[HKLM\Drivers\Display\SAMSUNG]

Table 10: LCD Driver: Registry path

Use the following parameters to configure the driver:

Key	Value	Meaning
"Mode"	Dword:	Number of the predefined configuration or new user configuration.
“EarlyLCDVoltage”	Dword:	Set this parameter to 1 power on the LCD voltage very early at the boot-process.

Table 11: LCD Driver: Registry

With parameter Mode you have the possibility to use one of the fixed configurations stored in the kernel or to define a new configuration in registry. Values between 0 and 99 are reserved for fixed configurations. For your own configuration you have to use values between 100 and 199.

Mode	Name	Resolution (wxh)	Type	Additional / Desc
0	Kyocera KCS3224	320x240	CSTN	NetDCU10
1	SHARP LM8V31	640x480	CSTN	NetDCU10
2	Toshiba LTM04C380K	640x480	TFT	NetDCU10
3	SHARP LQ104V1DG11	640x480	TFT	NetDCU10
4	SHARP LQ12S31	800x600	TFT	NetDCU10
5	SHARP LQ057Q3DC02	320x240	TFT	NetDCU10
6	Kyocera TCG057	320x240	TFT	NetDCU10
7	SHARP LQ057V3DG01	640x480	TFT	NetDCU10

Table 12: LCD Driver: Display modes NetDCU10

Mode	Name	Resolution (wxh)	Type	Clock (MHz)	Additional / Desc
0	VGA Standard Display	640x480	TFT	25	NetDCU9
1	SVGA Standard Display	800x600	TFT	40	NetDCU9
2	XGA Standard Display	1024x768	TFT	65	NetDCU9
3	SXGA Standard Display	1280x1024	TFT	108	NetDCU9
4	QVGA Standard Display	320x240	TFT	6,3	NetDCU9
5	XGA Standard Display	1024x768	TFT	56	NetDCU9

Table 13: LCD Driver: Display modes NetDCU9

For configurations with Mode higher than 99 you have to create a new sub-key with the Name ModeXXX. Detailed information how to perform these settings and a series of display drivers adjustments described in the documentation “NetDCU Display”.

6 Matrix-Keyboard

Implemented on NetDCU: 9,10

It is possible to connect a matrix keyboard to NetDCU. The organisation of this keyboard is 8 (rows) * 8 (columns) + 4 (static keys). So you can connect 64+4 keys to NetDCU. All inputs are internally connected with resistors to 3.3 Volt. Within the matrix you can only press one key at the same time. But it is possible to press a matrix key and one or more static keys at the same time. So it is possible to get the same behaviour as with a PC keyboard. The driver polls the keyboard every 20 ms. In the case a key is pressed, the driver reads the scan code and saves the value. After additional 20 ms it checks the scan code. If the scan code is unchanged the scan code will be transformed with the information stored in the mapping table in a PS2 keyboard scan code. The routing of this keyboard code is the same as the one from a PS2 keyboard. The mapping table for converting a scan code in an PS2 keyboard code is stored in the registry.

The settings which influence the driver are stored under key:

`[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX]`

Key	Value	Comment
Type	Dword:1	Details: Table 15: Matrix Keyboard: Type / Function
UseAsKey	Dword:<n>	This values specifies the port pins's which are used for keyboard input. See Table 20: Matrix Keyboard: Configuration J5 (1) – (3)
RowReverse	Dword:0	Reverse all bits of the row. Bit 0 to Bit 7, Bit 1 to Bit6
ColReverse	Dword:0	Reverse all bits of the column. Bit 0 to Bit 7, Bit 1 to Bit6
ChangeRowCol	Dword:0	Exchange the scan-value of row and column.
AutoKeyUp	Dword:0	If a matrix key is pressed and the previous key is not released, this value sends the KEYUP message to the system.
OutputScanCode	Dword:0	Set this value to 1 to output the scan-code of the currently pressed key as a debug message on the serial debug line.
StatKey1	Dword	Value for static key 1
StatKey2	Dword	Value for static key 2
StatKey3	Dword	Value for static key 3
StatKey4	Dword	Value for static key 4

Table 14: Matrix Keyboard: Registry

Type	Function
0	Matrix keyboard driver OFF
1	Matrix keyboard 8x8+4, 8 rows, 8 cols, 4 static keys, single key detection
3	Matrix keyboard 8x8, 8 rows, 8 cols, 0 static keys, single key detection
17	Matrix keyboard 8x8+4, 8 rows, 8 cols, 4 static keys, multiple key detection
19	Matrix keyboard 8x8, 8 rows, 8 cols, 0 static keys, multiple key detection

Table 15: Matrix Keyboard: Type / Function

Matrix keys are stored under:

```
[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX\MAP]
```

Static keys are stored under:

```
[HKLM\HARDWARE\DEVICEMAP\KEYBD\MATRIX]
```

Under `\MAP` you can make settings in the following form:

Key	Value
"1"	Dword:2
"2"	Dword:3
"3"	Dword:4
"4"	Dword:5

Table 16: Matrix Keyboard: Mapping matrix keys to PS-Codes

The value under Key (string!) is the scan code from the matrix keyboard. The range of this value is from 1 to 127 and must be given in decimal format. The value must be in hexadecimal form.
In the above example you send the PS2-Code 2 if you press the matrix key 1.

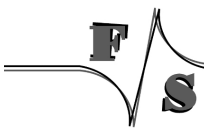
6.1 Default settings

Scan-Code	PS2-Code	V-Key
0x01	0x02	‚1‘
0x02	0x03	‚2‘
0x03	0x04	‚3‘
0x04	0x05	‚4‘
0x05	0x06	‚5‘
0x06	0x07	‚6‘
0x07	0x08	‚7‘
0x08	0x09	‚8‘
0x11	0x0A	‚9‘
0x12	0x0B	‚0‘
0x21	0xE04B	VK_LEFT
0x22	0xE04D	VK_RIGHT
0x23	0xE048	VK_UP
0x24	0xE050	VK_DOWN
0x25	0x01	VK_ESCAPE
0x26	0x1C	VK_RETURN

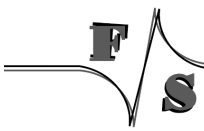
Table 17: Matrix Keyboard: Default settings

6.2 PS2 Scan-Codes

V-KEY	PS2-Scan-Code
0	// Scan Code 0x0
VK_ESCAPE	// Scan Code 0x1
'1'	// Scan Code 0x2
'2'	// Scan Code 0x3
'3'	// Scan Code 0x4
'4'	// Scan Code 0x5
'5'	// Scan Code 0x6
'6'	// Scan Code 0x7
'7'	// Scan Code 0x8
'8'	// Scan Code 0x9
'9'	// Scan Code 0xA
'0'	// Scan Code 0xB
VK_HYPHEN	// Scan Code 0xC
VK_EQUAL	// Scan Code 0xD
VK_BACK	// Scan Code 0xE
VK_TAB	// Scan Code 0xF
'Q'	// Scan Code 0x10
'W'	// Scan Code 0x11
'E'	// Scan Code 0x12
'R'	// Scan Code 0x13
'T'	// Scan Code 0x14
'Y'	// Scan Code 0x15
'U'	// Scan Code 0x16
'I'	// Scan Code 0x17
'O'	// Scan Code 0x18
'P'	// Scan Code 0x19
VK_LBRACKET	// Scan Code 0x1A
VK_RBRACKET	// Scan Code 0x1B
VK_RETURN	// Scan Code 0x1C
VK_LCONTROL	// Scan Code 0x1D
'A'	// Scan Code 0x1E
'S'	// Scan Code 0x1F
'D'	// Scan Code 0x20
'F'	// Scan Code 0x21
'G'	// Scan Code 0x22
'H'	// Scan Code 0x23
'J'	// Scan Code 0x24
'K'	// Scan Code 0x25
'L'	// Scan Code 0x26
VK_SEMICOLON	// Scan Code 0x27
VK_APOSTROPH	// Scan Code 0x28
VK_BACKQUOTE	// Scan Code 0x29
VK_LSHIFT	// Scan Code 0x2A
VK_BACKSLASH	// Scan Code 0x2B
'Z'	// Scan Code 0x2C
'X'	// Scan Code 0x2D
'C'	// Scan Code 0x2E
'V'	// Scan Code 0x2F
'B'	// Scan Code 0x30
'N'	// Scan Code 0x31
'M'	// Scan Code 0x32
VK_COMMA	// Scan Code 0x33



VK_PERIOD	// Scan Code 0x34
VK_SLASH	// Scan Code 0x35
VK_RSHIFT	// Scan Code 0x36
VK_MULTIPLY	// Scan Code 0x37
VK_LMENU	// Scan Code 0x38
VK_SPACE	// Scan Code 0x39
VK_CAPITAL	// Scan Code 0x3A
VK_F1	// Scan Code 0x3B
VK_F2	// Scan Code 0x3C
VK_F3	// Scan Code 0x3D
VK_F4	// Scan Code 0x3E
VK_F5	// Scan Code 0x3F
VK_F6	// Scan Code 0x40
VK_F7	// Scan Code 0x41
VK_F8	// Scan Code 0x42
VK_F9	// Scan Code 0x43
VK_F10	// Scan Code 0x44
VK_NUMLOCK	// Scan Code 0x45
VK_SCROLL	// Scan Code 0x46
VK_NUMPAD7	// Scan Code 0x47
VK_NUMPAD8	// Scan Code 0x48
VK_NUMPAD9	// Scan Code 0x49
VK_SUBTRACT	// Scan Code 0x4A
VK_NUMPAD4	// Scan Code 0x4B
VK_NUMPAD5	// Scan Code 0x4C
VK_NUMPAD6	// Scan Code 0x4D
VK_ADD	// Scan Code 0x4E
VK_NUMPAD1	// Scan Code 0x4F
VK_NUMPAD2	// Scan Code 0x50
VK_NUMPAD3	// Scan Code 0x51
VK_NUMPAD0	// Scan Code 0x52
VK_DECIMAL	// Scan Code 0x53
VK_SNAPSHOT	// Scan Code 0x54
VK_F11	// Scan Code 0x57
VK_F12	// Scan Code 0x58
VK_LWIN	// Scan Code 0x5B
VK_RWIN	// Scan Code 0x5C
VK_APPS	// Scan Code 0x5D
VK_HELP	// Scan Code 0x63
VK_F13	// Scan Code 0x64
VK_F14	// Scan Code 0x65
VK_F15	// Scan Code 0x66
VK_F16	// Scan Code 0x67
VK_F17	// Scan Code 0x68
VK_F18	// Scan Code 0x69
VK_F19	// Scan Code 0x6A
VK_F20	// Scan Code 0x6B
VK_F21	// Scan Code 0x6C
VK_F22	// Scan Code 0x6D
VK_F23	// Scan Code 0x6E
VK_F24	// Scan Code 0x76
VK_DIVIDE	// Scan Code 0xE035
VK_SNAPSHOT	// Scan Code 0xE037
VK_RMENU	// Scan Code 0xE038
VK_HOME	// Scan Code 0xE047
VK_UP	// Scan Code 0xE048



VK_PRIOR	// Scan Code 0xE049
VK_LEFT	// Scan Code 0xE04B
VK_RIGHT	// Scan Code 0xE04D
VK_END	// Scan Code 0xE04F
VK_DOWN	// Scan Code 0xE050
VK_NEXT	// Scan Code 0xE051
VK_INSERT	// Scan Code 0xE052
VK_DELETE	// Scan Code 0xE053
VK_LWIN	// Scan Code 0xE05B
VK_RWIN	// Scan Code 0xE05C
VK_APPS	// Scan Code 0xE05D

Table 18: Matrix Keyboard: PS2 Scan-Codes

6.3 Scan-Code Matrix 8x8

	C0	C1	C2	C3
R0	0x01	0x02	0x03	0x04
R1	0x11	0x12	0x13	0x14
R2	0x21	0x22	0x23	0x24
R3	0x31	0x32	0x33	0x34
R4	0x41	0x42	0x43	0x44
R5	0x51	0x52	0x53	0x54
R6	0x61	0x62	0x63	0x64
R7	0x71	0x72	0x73	0x74

	C4	C5	C6	C7
R0	0x05	0x06	0x07	0x08
R1	0x15	0x16	0x17	0x18
R2	0x25	0x26	0x27	0x28
R3	0x35	0x36	0x37	0x38
R4	0x45	0x46	0x47	0x48
R5	0x55	0x56	0x57	0x58
R6	0x65	0x66	0x67	0x68
R7	0x75	0x76	0x77	0x78

Table 19: Matrix Keyboard: Scan-Code Matrix 8x8

Note:

See see next table for a assignment between R7..R0 and C7..C0 and physical pins.

6.4 Port configuration on J5

Pin	2	3	4	5	6	7	8	9
R/W	W	W	W	W	W	W	W	W
Meaning	R7	R6	R5	R4	R3	R2	R1	R0
UseAsKey Bit	7	6	5	4	3	2	1	0

Table 20: Matrix Keyboard: Configuration J5 (1)

Pin	---	---	---	---	10	11	13	15
R/W	---	---	---	---	R	R	R	R
Meaning					C8	C9	C10	C11
UseAsKey Bit	15	14	13	12	11	10	9	8

Table 21: Matrix Keyboard: Configuration J5 (2)

Pin	17	18	19	20	21	22	23	24
R/W	R	R	R	R	R	R	R	R
Meaning	C7	C6	C5	C4	C3	C2	C1	C0
UseAsKey Bit	23	22	21	20	19	18	17	16

Table 22: Matrix Keyboard: Configuration J5 (3)

6.5 Matrix Keyboard connector J5

Pin	Signal	Function ColReverse=0	Function ColReverse=1
1	---		
2	GPIO7	R7	R7
3	GPIO6	R6	R6
4	GPIO5	R5	R5
5	GPIO4	R4	R4
6	GPIO3	R3	R3
7	GPIO2	R2	R2
8	GPIO1	R1	R1
9	GPIO0	R0	R0
10	GPIO9	StatKey1	StatKey1
11	GPIO10	Statkey2	StatKey2
12	---		
13	GPIO11	StatKey3	StatKey3
14	---		
15	GPIO12	StatKey4	StatKey4
16	GND		
17	KBIN0	C7	C0
18	KBIN1	C6	C1
19	KBIN2	C5	C2
20	KBIN3	C4	C3

21	KBIN4	C3	C4
22	KBIN5	C2	C5
23	KBIN6	C1	C6
24	KBIN7	C0	C7
25	V_{CC}		
26	V_{DD}		

Table 23: Matrix Keyboard: connector J5

7 NANDFMD Driver

Implemented on NetDCU: 9,10

```
[HKLM\Drivers\BuiltIn\NandFmd]
```

Required settings:

Key	Value	Comment
"Prefix"	“DSK“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“NANDFMD.DLL“	name of the DLL with the driver
“Order“	Dword:0	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
“Index“	Dword:1	This value specifies the device index, a value from 0 through 9.
“FriendlyName“	"NetDCU flash file system driver“	
“Profile“	“FFSDISK“	Drive name
“FSD”	“FATFS.DLL”	
Flags	Dword:0x1000	

Table 24: NANDFMD: Registry

```
[HKEY_LOCAL_MACHINE\Loader]
      "SystemPath"=multi_sz:"\\ffsdisk\\"
```

Required settings:

Key	Value	Comment
"SystemPath"	multi_sz:"\\ffsdisk\\"	Extends the search path to the folder FFSDISK.

Table 25: NANDFMD: extending the search path

8 FS-Bus Driver

Implemented on NetDCU: 9,10

This driver is needed to access the parallel extension bus “FS-BUS” at Connector J4. This driver is also the base driver for all available interfaces for NetDCU.

The registry key for the driver is:

```
[HKLM\Drivers\BuiltIn\PARINTF]
```

Required settings:

Key	Value	Comment
"Prefix"	“PIF“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“PARINTF.DLL“	Name of the DLL with the driver.
“Order“	Dword:0x10	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
“Index“	Dword:1	This value specifies the device index, a value from 0 through 9.
“Ioctl“	Dword:4	Call post-initialisation function.
“FriendlyName“	"Parallel interface driver for NetDCU“	
“WaitStates”	DWORD:	Table 27: FS-Bus: Waitstates NetDCU9 Table 29: FS-Bus: Waitstates NetDCU10
“Debug”	DWORD:0	0: no debug output 4: output values read from registry 5: output more debug messages
“IRQ”	DWORD:	NetDCU9: 0x85 NetDCU10: 0x21
“IRQCfg”	DWORD:2	0=none, 1=raising edge, 2=falling edge

Table 26: FS-Bus: Registry

8.1 Waitstates NetDCU9

MSCn	Bit	Description	Default
RBUFFx	31	Return Data Buffer vs. Streaming Behavior When slow memory devices are used in the system (VLIO, slow SRAM/ROM), this bit must be cleared to allow the system to process other information rather than remain idle while all the data is read from the device. When this bit is cleared, the system is allowed to process other information. When the bit is set, the system halts and waits until all data is returned from the device. When synchronous static memory devices have been enabled for a given bank, this bit defaults to streaming behavior. The register bit is read as 0 unless it has specifically been programmed to a 1. This behavior cannot be overridden. 0 = Slower device (return data buffer) 1 = Faster device (streaming behavior)	0
RRRx	[30:28]	ROM/SRAM Recovery Time The value of this bit is half the number of memory clock cycles from the time that chip select is de-asserted after a read or write until the next chip select (of a different static memory bank) or nSDCS is asserted. This field must be programmed with the highest of the three values: tOFF divided by two, write pulse high time (flash memory/SRAM), and write recovery before read (flash memory). tOFF = RRRx * 2 + 1 **NOTE:** The MSCx[RRR] value (recovery time after chip select deasserted) must be reprogrammed prior to switching the processor to deep-idle mode to avoid long times when the MD	0
RDNx	[27:24]	ROM Delay Next Access The RDN field is encoded as follows: ENCODED (Programmed) Value -----> DECODED (Actual) Value 0-11 -----> 0-11 12 -----> 15 13 -----> 20 14 -----> 26 15 -----> 30 Use the DECODED value in the equations below for RDNx instead of the actual RDNx value. For burst ROM or flash memory: • RDNx + 1 = number of MEM_CLKs from Address to Data Valid for subsequent access. For flash memory or SRAM: • RDNx + 1 = number of CLK_MEMs nWE is asserted for write accesses For VLIO: • RDNx * 2 = amount of time nOE or nPWE is deasserted to address hold and address setup to nOE or nPWE assertion time. **NOTE:** For VLIO, this number must be greater than or equal to 2. The memory	0

MSCn	Bit	Description	Default
RDFx	[23:20]	ROM Delay First Access The encoding scheme is: ENCODED (programmed) value ------> DECODED (actual) value: 0-11 ------> 0-11 12 ------> 15 13 ------> 20 14 -----> 26 15 ------> 30 The DECODED value represents: • Number of memory-clock cycles (minus 2) from address to data valid for first read access from all devices except VLIO • Number of memory clock cycles (minus 1) from address to data valid for subsequent read accesses to non-burst devices except VLIO • Number of memory clock cycles (minus 1) of nWE assertion for write accesses to all types of flash memory For variable-latency I/O, this determines the minimum number of memory clock cycles (minus 1) of nOE (nPWE) assert time for each beat of read (write). **NOTE:** For VLIO, this number must be greater than or equal to 3. The memory	0

Table 27: FS-Bus: Waitstates NetDCU9

Note:

Set up the WaitStates value starting at the LSB of the dword (Registry).

Name	RBUFFx	RRRx			RDNx				RDFx			
Bitpos WaitStates value	11	10	9	8	7	6	5	4	3	2	1	0
Bitpos MSCn	31	30	29	28	27	26	25	24	23	22	21	20

Table 28: FS-Bus: Set up WaitStates value NetDCU9

8.2 Waitstates NetDCU10

BANKCONn	Bit	Description	Default
Tacs	[14:13]	Address set-up time before nGCSn 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	00
Tcos	[12:11]	Chip selection set-up time before nOE 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	00
Tacc	[10:8]	Access cycle 000 = 1 clock 001 = 2 clocks 010 = 3 clocks 011 = 4 clocks 100 = 6 clocks 101 = 8 clocks 110 = 10 clocks 111 = 14 clocks Note: When nWAIT signal is used, Tacc >= 4 clocks.	111

BANKCONn	Bit	Description	Default
Tcoh	[7:6]	Chip selection hold time after nOE 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	000
Tcah	[5:4]	Address hold time after nGCSn 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	00

Table 29: FS-Bus: Waitstates NetDCU10

Programming Example:

Headerfile:

```
#include <parintfsdk.h>
```

Listing 15: FS-Bus: Headerfile

A. Opening FS-Bus handle

```
HANDLE hPIF = CreateFile(L"PIF1:",GENERIC_WRITE, 0, NULL, OPEN_EXISTING
                                 ,FILE_ATTRIBUTE_NORMAL,NULL );

if(INVALID_HANDLE_VALUE == hPIF)
{
  ERRORMSG(1,(L"INVALID HANDLE VALUE\r\n"));
  return -1;
}
```

Listing 16: FS-Bus: Opening handle

B. Write data without address

```
unsigned char data[3] = { 0x1, 0x2, 0x3 };
DWORD dwBytesWrite = 1;
WriteFile( hDIO, data, 3, &dwBytesWrite, NULL );
if( dwBytesWrite != 3 )
{
   ERRORMSG(1,L"Can not write to PIF1. LE = 0x%x\r\n",GetLastError()));
}
```

Listing 17: FS-Bus: Write data without address

C. Read data without address

```
unsigned char data[3] = { 0x1, 0x2, 0x3 };
DWORD dwRead;
ReadFile( hDIO, data, 3, &dwRead, NULL );
if( dwRead != 3 )
{
   ERRORMSG(1,L"Can not read from PIF1. LE = 0x%x\r\n",GetLastError()));
}
```

Listing 18: FS-Bus: Read data without address

D. Write address / data to bus

```
PARINTFRW cData;
DWORD dwBytesReturend;
cData.chAddress = 0x20;
```

```
if( !DeviceIoControl( m_hPIF,
     IOCTL_PARINTF_WRITE,
     &cData, sizeof(cData), NULL, 0,
     &dwBytesReturend, NULL ) )
{
   ERRORMSG(1,L"Can not write to PIF1. LE = 0x%x\r\n",GetLastError()));
}
```

Listing 19: FS-Bus: Write address / data to bus

E. Read one byte from bus

```
PARINTFRW cData;
DWORD dwBytesReturend;
cData.chAddress = 0x20;
cData.chData = 1;

if( !DeviceIoControl( m_hPIF,
     IOCTL_PARINTF_READ,
     &cData, sizeof(cData), NULL, 0,
     &dwBytesReturend, NULL ) )
{
   ERRORMSG(1,L"Can not read from PIF1. LE = 0x%x\r\n",GetLastError()));
}
```

Listing 20: FS-Bus: Read one byte from bus

F. Handle interrupt

```
DWORD dwTimeOut = 30000;      //Timeout to be blocked by _WAIT_IRQ (DeviceIoControl).
DWORD dwWaitRes = -1;         //Return value that indicates the event result.
                              //WAIT_OBJECT_0,WAIT_ABANDONED,WAIT_TIMEOUT.
                              //For more information see WaitForSingleObject API.
DWORD dwBytesReturend = -1;
//Wait for interrupt
DeviceIoControl( hPIF,IOCTL_PARINTF_WAIT_IRQ, &dwTimeOut, sizeof(DWORD)
                                            , &dwWaitRes, sizeof(DWORD)
                                            , &dwBytesReturend, NULL );
//Call InterruptDone
DeviceIoControl( hPIF,IOCTL_PARINTF_DONE_IRQ, NULL, sizeof(DWORD), NULL, sizeof(DWORD)
                                            , &dwBytesReturend, NULL );
```

Listing 21: FS-Bus: Handling interrupt

9 Serial Driver

Implemented on NetDCU: 9,10

This driver is needed to access the serial interfaces `COM1:`, `COM2:` and `COM3:`.

The registry key for the driver is:

```
[HKLM\Drivers\BuiltIn\SERIAL1]
[HKLM\Drivers\BuiltIn\SERIAL2]
[HKLM\Drivers\BuiltIn\SERIAL3]
```

Optional settings:

Key	Value	Comment
"Priority256"	Dword:104	Priority for serial receive/transmit thread. Default: 104
“WaterMarker“	Dword:1,8,16,32	FIFO trigger level. Default: 8
RS485	Dword:	Enable RS485 mode for COM2: Default: 0 (NetDCU10)

RS485 Mode

With NetDCU10 you can toggle COM2: between RS232 and RS485. To do this, you have to add registry value RS485 and set it to 1. Additionally you have to modify DCB in the way that dcb.fRtsControl is equal to RTS_CONTROL_TOGGLE.

Programming Example:

```
DCB dcb;

dcb.DCBlength = sizeof( dcb );
GetCommState( g_hCOM, &dcb );

dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;
dcb.BaudRate = CBR_38400;
dcb.fOutxCtsFlow = 0;
dcb.fRtsControl = RTS_CONTROL_TOGGLE;
SetCommState( g_hCOM, &dcb );
```

Listing 22: Serial: Using COM2 in RS485 mode (NetDCU10)

10 Driver for NDCU-ADP/UART

Implemented on NetDCU: 9,10

NetDCU has 3 integrated serial lines. If more serial lines or a serial line with all modem signals or a RS422/485 is needed you can use the extension board NDCU-ADP/UART to get two more serial lines. For hardware documentation take a look at document “UART-Adapter”.

This driver can only work if the driver PARINTF is installed and running. Check this under `\HKLM\Drivers\Active`.

Installation of the driver is done by setting some registry values under the following registry key:

```
[HKLM\Drivers\BuiltIn\SERIAL3]
[HKLM\Drivers\BuiltIn\SERIAL4]
```

Key	Value	Comment
"Prefix"	“COM“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“UARTINTF.DLL“	Name of the DLL with the driver.
“Order“	Dword:35 Dword:36	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
“Index“	Dword:3 Dword:4	This value specifies the device index, a value from 0 through 9.
“Ioctl“	Dword:4	Call post-initialisation function.
“Port“	“PIF1:”	This value specifies the device name of that parallel port. Normally this is PIF1:.
DeviceArrayIndex	Dword:0 Dword:1	Number of the hardware port you want to access. 0:Port at connector J1 1: Port at connector J2
RcvrTriggerLevel	Dword:8	The UART on the NDCU-ADP/UART has a 16 byte receive FIFO. With this value you can setup the trigger level for the receive interrupt.
Priority256	Dword:103	Priority for this thread.
“FriendlyName“	“Serial driver for NetDCU“	

Table 30: Registry: NDCU-ADP/UART

11 Driver for NDCU-ADP/CAN2

Implemented on NetDCU: 9,10

The connectivity of NetDCU could be significantly enhanced with 2 CAN lines by usage of NDCU-ADP/CAN2. For hardware documentation take a look at document “CAN-Adapter”. This document only describes the installation of the driver. The usage of the driver is described in document “WINCE-CAN-Interface”

This driver can only work if the driver PARINTF is installed and running. Check this under `\HKLM\Drivers\Active`.

Installation of the driver is done by setting some registry values under the following registry key:

```
[HKLM\Drivers\BuiltIn\CAN1]
[HKLM\Drivers\BuiltIn\CAN2]
```

Key	Value	Comment
Prefix	“CID“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
Dll	“CANINTF.DLL“	Name of the DLL with the driver.
Order	Dword:75 Dword:76	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
Index	Dword:0 Dword:1	This value specifies the device index, a value from 0 through 9.
Ioctl	Dword:4	Call post-initialisation function.
Port	“PIF1:”	This value specifies the device name of the parallel port. Normally this is PIF1:.
DeviceArrayIndex	Dword:0 Dword:1	Number of the hardware port you want to access. 0:Port at connector J3 1: Port at connector J4
Priority256	Dword:103	Priority for this thread.
FriendlyName	“CAN driver for NetDCU“	
Baudrate	Dword:1000000	
Virtualize	Dword:0	
“Debug”	Dword:0|4	Set to 4 to get list of registry settings at serial debug port. Default: 0

Table 31: Registry: NDCU-ADP/CAN2

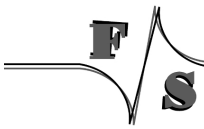
12 Touchpanel Driver

Implemented on NetDCU: 9,10

[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH]

Key	Value	Comment
CalibrationData	"0,0,0,0,0,"	Set this value to the given string to avoid the calibration screen after restart.
MinCalibrationPointCount	Dword	Min. amount of samples per calibration point. The value should be high enough to prevent spurious touches, but low enough that the user doesn't have to hold the pen on each crosshair too long. Default: 10
TouchSamples	Dword:3...20	With this value you can adjust the amount of samples that are used to create the position value. As more samples as longer the time you have to press on the same place. Default: 7
SamplePeriodLowHns	Dword	Sample period settings in 100 ns units for low sample periods. Default: 20ms (200000)
SamplePeriodHighHns	Dword	Sample period settings in 100 ns units for high sample periods. Default: 10ms (100000)
PollUpTimeMultiplier	Dword: 1…20	Poll time in pen up state is PollUpTimeMultiplier *SamplePeriodXXXHns. So a short PollUpTimeMultiplier causes a faster first touch detection. Default: 10 (NetDCU9 only, since V1.4)
DeltaXCoordTolerance	Dword:0...0x3FF	This value is used by the touch sample filter routine to accept and reject points. Increasing the tolerance will generally allow faster pen movements to be detected. This will also increase noise and tend to cause erratic touch behavior. Default: 20
DeltaYCoordTolerance	Dword:0...0x3FF	This value is used by the touch sample filter routine to accept and reject points. Increasing the tolerance will generally allow faster pen movements to be detected. This will also increase noise and tend to cause erratic touch behavior. Default: 16
MinMove	Dword:1...0x3FF	Minimum move (A/D resolution) before MouseMove is signaled. Default: 5

Key	Value	Comment
MaxMove	Dword:1...0x3FF	Maximum move (A/D resolution) which is recognized and send to application layer. Default: 50
AutoCalib	Dword:0...10000	Time in ms before event 20 is signaled to application layer when touch is pressed. Can be used for automatic touch calibration. AutoCalib=0 disables this function. Default: 0
Debug	Dword:0|4	Set to 4 to get list of registry settings at serial debug port. Default: 0
UseStandardDeviation	Dword:0…100	Value in percentage of the touch controller resolution. If standard deviation of the "TouchSamples" values exceeds this value the data colletion is marked as corrupt. In contrast to the "DeltaX/YCoordTolerance" we will include several runaway values but exclude high statistical spreaded values. Standard deviation is checked before "DeltaX/YCoordTolerance". Default: 0, means standard deviation is not used. Supported only on NetDCU10.
CheckDownWhileSample	Dword:0|1	1 = Marking data collection as corrupt when "touch up" is detected while sampling "TouchSamples". Value supported only on NetDCU10 (NetDCU9 driver does always check it). Default: 0
UseStallCounter	Dword:0|1	You can decide if you want to stall execution during AdcReadHoldOff or if you want to sleep for at least one ms. Default: 1
Debounce	Dword:0|1	Enables debouncing. Debouncing marks a sample as valid if the previous sample has the same value. Default: 1
DebounceJitter	Dword:0…0x3FF	Filters some jitter from the samples. Default: 3 -> LSB1,2 are filtered.
DebounceRepeatHns"	Dword	If the two samples for debouncing are different, than we adjust the measuring interval to this value. Default: 5ms



Key	Value	Comment
AdcReadHoldoffHns”	Dword	Amount of time (in100 ns units) to wait after biasing the plates before starting an ADC read to determine an X or Y coordinate. This allows the voltage at the ADC input to settle. More time may be needed if large capacitors or other filtering devices are used. Wait times that are too small will result in poor touch performance (unstable pen position). Wait times that are too long will cause poor system performance and may reduce the touch sampling frequency.

Table 32: Touch: Registry

[HKEY_LOCAL_MACHINE\SYSTEM\CALIBRUI]

Possible settings:

Key	Value	Comment
"NoKeyboard"	Dword:1	This parameter tells touch panel calibration to not wait for a keystroke at the end of calibration.

Table 33: Touch: CALIBRUI

[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH]

Possible settings:

Key	Value	Comment
"Priority256" ”HighPriority256“	Dword:109	Set this value to adjust the priority of the touch panel driver.

Table 34: Touch: Adjust priority

13 USB Mass Storage Class Driver

Implemented on NetDCU: 9,10

```
[HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\Mass_Storage_Class\6]
```

Possible settings:

Key	Value	Comment
"Prefix"	“DSK“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“USBDISK6.DLL“	name of the DLL with the driver
“Folder“	“USB Disk”	
“Ioctl“	Dword:4	Call post-initialisation function.
“FSD”	“FATFS.DLL”	
“DisableModeSense10”	DWORD:0	Disable the call to the function which detects the device mode. Necessary for some USB memory adaptors.

Table 35: USB Maß Storage: Registry

14 Soft-Keyboard

Implemented on NetDCU: 9,10

Sometimes it is useful to have a virtual keyboard on your display which can be controlled by using the touch panel.

The configuration program NDCUCFG (version 012 and higher) has a command to show the input panel on the screen (sip on).

Installation of the driver softkb.dll is done by setting some registry values under the following registry key:

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\SIP]

Required settings:

Key	Value	Comment
"Prefix"	“SIP“	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“Dll“	“SOFTKB.DLL“	name of the driver file
“Order“	Dword:1	This value specifies the load order for the driver. If two drivers have the same load order value, the drivers load in the order that they occur in the registry.
“Index“	Dword:0	This value specifies the device index, a value from 0 through 9.
Flags	Dword:0x10	

Table 36: SIP: Registry

15 Audio Driver

Implemented on NetDCU: 10

Audio driver for NetDCU is implemented as wavdev2 driver and can be configured under the following registry key:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio]
```

Possible settings:

Key	Value	Comment
“Prefix”	“WAV”	This required value specifies the driver’s device file name prefix. It is a three-character identifier, such as COM.
“DLL”	“wavedev_n10.dll”	name of the driver file
“Order”	Dword:0x20	
“Index”	Dword:1	This value specifies the device index, a value from 0 through 9.
“InChannel”	Dword:n	This value selects the input channel. 2 = Line-In 3 = Microphone
“MicBoost”	Dword:0|1	Set this 1 to boost microphone input by 20dB. Default: 0
“LineInOutBypass”	Dword:0|1	Set this to 1 to route Line-In directly to Line-Out. Default: 0
“LineInVolLeft”	Dword:n	Volume for Line-In left. Default: 0x17
“LineInVolRight”	Dword:n	Volume for Line-In right. Default: 0x17
“Debug”	Dword:0|4	Set to 4 to get list of registry settings at serial debug port. Default: 0

Table 37: Audio: Registry

16 NDCUCFG

Implemented on NetDCU: all

This utility is always included in the Windows CE image and enables the customer to access the registry from the command line and to call some additional helper functions.

Ndcucfg.exe can be started over serial, telnet or from within a command window.

By default, ndcucfg.exe is started from a Launch/Depend configuration in

```
[HKEY_LOCAL_MACHINE\Init]
```

and receives commands over serial line COM1:. If you want to change the serial line you can find settings of *ndcucfg.exe* under the following registry key:

```
[HKEY_LOCAL_MACHINE\System\NDCUCFG]
```

Possible settings:

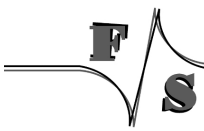
Key	Value	Comment
"Port"	“COM1:”	NDCUFG is automatically started during boot because of a entry in HKLM\INIT. With this value you can specify on which serial line ndcucfg uses for communication.
"BatchFile“	String	The commands in the file will be executed during start of ndcucfg.exe.

Table 38: NDCUCFG: Registry

List of commands (not complete):

- *display mode set <mode>*
  Changes the display mode to the given number.
- *display mode get*
  Retrieves the display mode.
- *display rotate get*
  Retrieves the display rotation angle.
- *display rotate set <n>*
  Changes the display rotation to the given angle.
- *reg open*
  opens the root key under HKLM
- *reg open <key>*
  opens the specified key under HKLM(open)
- *reg opencu <key>*
  opens the specified key under HKCU(opencu)
- *reg enum*
  displays a list of all keys and values under the current location
- *reg set value <name> dword <value>*
- *reg set value <name> string <value>*

- *reg set value <name> multi <value1>;<value2>*
- *reg set value <name> hex <value>,<value>,<value>*
  sets/creates the value with name <name> to the value <value>
- *reg create key <name>*
  Creates the specified sub-key and opens it.
- *reg del value <name>*
  Delete the specified value from registry.
- *reg del key <name>*
  Delete the specified key from registry.
- *reg save*
  Saves the registry in flash memory, so that modifications are available after reset.
- *fat format <volume>*
  Formats the volume with name <volume>.
- *contrast +*
  Increase contrast voltage of LCD (small steps)
- *contrast ++*
  Increase contrast voltage of LCD (large steps)
- *contrast -*
  Decrease contrast voltage of LCD (small steps)
- *contrast --*
  Decrease contrast voltage of LCD (large steps)
- *contrast get*
  Returns the current contrast voltage of LCD.
- *contrast set <n>*
  Sets the contrast voltage of LCD. The value is the high time for the PWM circuit.
- *backlight on*
  Switch on backlight of LCD
- *backlight off*
  Switch off backlight of LCD
- *touch calibrate*
  Shows the calibration screen for the touch panel.
- *sip on*
  Shows the input panel window.
- *sip off*
  Hides the input panel window.
- *reboot*
  Reboots the device.
- *cert import cert <store> <file>*
  Import certificate with filename <file> into certificate store <store>. Values for <store> MY, CA or ROOT
- *cert import pkey <store> <file>*
  Import private key from file into certificate store MY, CA or ROOT
- *cert enum*
  List all certificates from store MY, CA and ROOT
- *cert delete <store> <store name>*
  Delete certificate
- *user create <name> <password>*
  Creates new use with password
- *user delete <name>*
  Delete user
- *user enum*
  List all users
- *REM <comment>*
  Records comments (remarks) in a batch file.



- *ECHO <message>*
  Displays messages.
- *start <file name> <parameter>*
  Creates a new process and its primary thread.
- *ndcucfg -B<file name>*
  runs <file name> as batch process.

17 Extending the Search Path

Implemented on: all

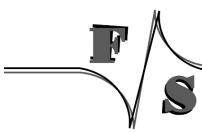
It's possible to extend the default path that the kernel uses to locate executable files. The necessary entry can be found under registry key:

```
HKEY_LOCAL_MACHINE\Loader
```

Possible settings:

Key	Value	Comment
"SystemPath"	Multi:"\\ffsdisk\\"	To extend the path you must add values to the value.

Table 39: Ext. Search Path



18 Module FATUI

This module implements the user interface for the FAT file system. This module is used if the file system must show a dialog box to interact with the user. This happens for example if the user inserts a PCMCIA card or the file system detects an unformatted drive.
In general dialog boxes are a good idea. But if the user has not connected a display to device all kind of display output is unusable. Because of that we have added an option to redirect all display output to the serial line at connector ST6. The value can be found under key:

```
[HKLM\System\FATUI]
```

Parameter:

Key	Value	Meaning
"AutoAnswer"	Dword:0|1	Set this value to 1 to redirect all output to serial line at connector ST6. The question to format the drive will be automatically answered with YES, all other questions will be answered with FATUI_NONE.

Table 40: FATUI: redirect output to serial line

19 Module NETUI

This module implements the user interface for the Network access. This module is used if a network resource is accessed which needs a user and password. By setting the described parameters, it is possible to avoid the normally shown dialog box.

The value can be found under key:

```
[HKLM\System\NETUI]
```

Parameter:

Key	Value	Meaning
"AutoLogon"	Dword:0|1	Set this value to 1 to use the registry values UserName and Password for network access.
"UserName“	String	
"Password“	String	

Table 41: NETUI: AutoLogon

20 Appendix

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. F&S Elektronik Systeme assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained in this documentation.
F&S Elektronik Systeme reserves the right to make changes in its products or product specifications or product documentation with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

F&S Elektronik Systeme makes no warranty or guarantee regarding the suitability of its products for any particular purpose, nor does F&S Elektronik Systeme assume any liability arising out of the documentation or use of any product and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Products are not designed, intended, or authorized for use as components in systems intended for applications intended to support or sustain life, or for any other application in which the failure of the product from F&S Elektronik Systeme could create a situation where personal injury or death may occur. Should the Buyer purchase or use a F&S Elektronik Systeme product for any such unintended or unauthorized application, the Buyer shall indemnify and hold F&S Elektronik Systeme and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that F&S Elektronik Systeme was negligent regarding the design or manufacture of said product.

Listings

Figures

Tables