

Pfaadt Software

Ingenieurbüro Hartmut & Klaus Pfaadt GbRmbH

Paul-Gerhard-Weg 8
D-79183 Waldkirch

info@pfaadtsoft.de
www.pfaadtsoft.de

Tel. +49 7681 / 493 997 2
Fax +49 7681 / 493 997 8

Embedded Software, Windows CE.NET,
BIOS Adaptations, Driver Development,
Realtime Programming, Applications

Philips SJA1000 CAN Bus Driver for Windows CE

Copyright © 2006 Pfaadt Software

Author: Hartmut Pfaadt
Date: 02/28/2006
Revision: 1.4

Table of Contents

1. Introduction	3
2. Realization	4
Starting the driver	4
The sample test application can_api_test.exe.....	5
CAN Interfaces.....	7
Data Flow und Events	8
CAN Driver Interface CAN_API.H	9
3. Registry Configuration.....	11

1. Introduction

CAN controllers are small and smart microprocessors that take care about receiving, transmitting and error handling independent from the CPU. The CPU sends requests to the can controller that will be processed and reported back to the CPU by using interrupts. The communication to the can controller will be done with shared memory accesses (memory or i/o based).

Requirements:

- Encapsulation of the access functions in a separate driver
- Realtime: Collection of data from the can controller in sufficient time
- Easy to use functions for can transfer, error handling and configuration

2. Realization

Starting the driver

The CAN driver `can_driver.dll` uses the Windows CE stream interface. This interface offers common and easy to use functions like `init`, `open`, `read`, `write` and `ioctl`.

The CAN driver will be initialized at Windows CE startup with the following registry keys:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN1]
  "Prefix"="CAN"
  "Dll"="can_driver.dll"
  "Order"=dword:0
  "Index"=dword:1
```

In this case the CAN driver uses the device name „CAN1:“.

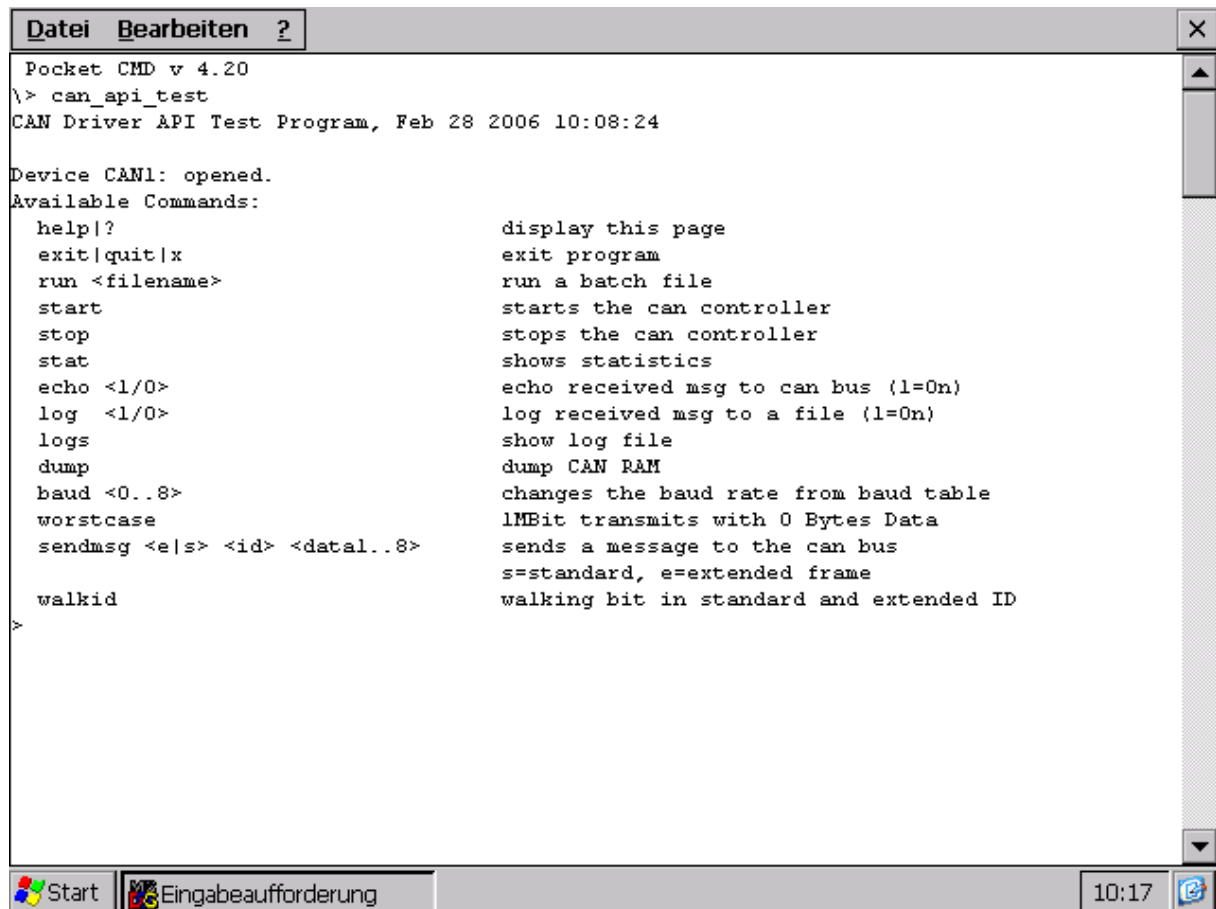
After starting the driver, the CAN Controller chip will be first configured. The CAN interrupts are initialized, but the CAN Controller is still in Reset Mode and not active to the CAN bus.

You need a few more keys depending on your hardware. See Chapter “Registry Configuration” for more information.

Add the following line to the modules section of your `platform.bib`:

```
can_driver.dll      $_FLATRELEASEDIR\can_driver.dll    NK SH
can_api_test.exe   $_FLATRELEASEDIR\can_api_test.exe  NK
```

The sample test application `can_api_test.exe`



```
Pocket CMD v 4.20
\> can_api_test
CAN Driver API Test Program, Feb 28 2006 10:08:24

Device CAN1: opened.
Available Commands:
  help|?          display this page
  exit|quit|x     exit program
  run <filename>  run a batch file
  start           starts the can controller
  stop            stops the can controller
  stat           shows statistics
  echo <l/0>      echo received msg to can bus (l=0n)
  log <l/0>       log received msg to a file (l=0n)
  logs           show log file
  dump           dump CAN RAM
  baud <0..8>    changes the baud rate from baud table
  worstcase      lMBit transmits with 0 Bytes Data
  sendmsg <e|s> <id> <data1..8> sends a message to the can bus
                                     s=standard, e=extended frame
  walkid         walking bit in standard and extended ID

y
```

CAN_API_TEST.EXE is a easy sample application to demonstrate how to use the `can_driver` with a user application. This application first opens the first instance of the device CAN (“CAN1:”).

The console application `can_api_test` uses a simple menu to test the API functions (also see source `main.cpp`).

To work with a secondary CAN Controller, start “`can_api_test can2:`”.

Command description:

Start	Starts the CAN Controller. You are now active on the can bus.
Stop	Stops the CAN Controller. Switches back to reset/configuration mode.
Stat	Displays some Statistics
Echo 1	sends all received messages back to the can bus.
Echo 0	echo mode is off.
Log 1	Stores all received Messages to a file.
Log 0	Log-Mode is off.
Logs	shows the log file
Dump	shows the can ram
Baud 0..x	switches the baud rate according to the baudrate table entry
Worstcase	sends smallest and fastest can messages on the can bus
Sendmsg	sends a message to the can bus "sendmsg s 123 1 2 3 4 5 6" sends a standard frame (11bit) with ID 123 and 6 bytes of data. "sendmsg e 123" sends an extended frame (29bit) with ID 123 and 0 bytes of data.
Walkid	walking bit on the ID for testing (e.g. acceptance filter).

You can run the commands as a batch file.

Sample Batch send.bat:

```
Start
Sendmsg s 123 1 2 3 4 5 6
Sendmsg e 123
Stat
Dump
Stop
Run
```

Start the batch with "run send.bat"

CAN Interfaces

CAN_API.H

Functions used by the application

```
BOOL  CAN_StartChip  (HANDLE hDevice);
BOOL  CAN_StopChip  (HANDLE hDevice);
BOOL  CAN_GetNextReceivedFrame(HANDLE hDevice, MessageFrame *RxFrameBuffer);
BOOL  CAN_SendFrame  (HANDLE hDevice, MessageFrame *TxFrameBuffer);
BOOL  CAN_GetError   (HANDLE hDevice, DWORD *ErrorPtr);
BOOL  CAN_SetBaudRate(HANDLE hDevice, BYTE *index);
DWORD CAN_GetCANMemorySnapShot(HANDLE hDevice, BYTE *ptr, DWORD size);

#ifdef HCAN2
#define SINGLE_FILTER_MODE 0x01
#define DUAL_FILTER_MODE  0x00
BOOL  CAN_SetGlobalAcceptanceFilter(HANDLE hDevice, BYTE *AcceptanceFilter, BYTE size);
#endif
```

CAN_IOCTL.H

IO-Controls for communication between CAN-API and the can driver.

```
// create unique system I/O control codes (IOCTL)

IOCTL_START_CHIP,
IOCTL_STOP_CHIP,
IOCTL_RECEIVE_MESSAGE,
IOCTL_SEND_MESSAGE,
IOCTL_GET_ERROR,
IOCTL_SET_BAUDRATE,
IOCTL_SET_ACCEPTANCE_FILTER,
IOCTL_GETCANMEMORYSNAPSHOT
```

CAN_DRIVER.H

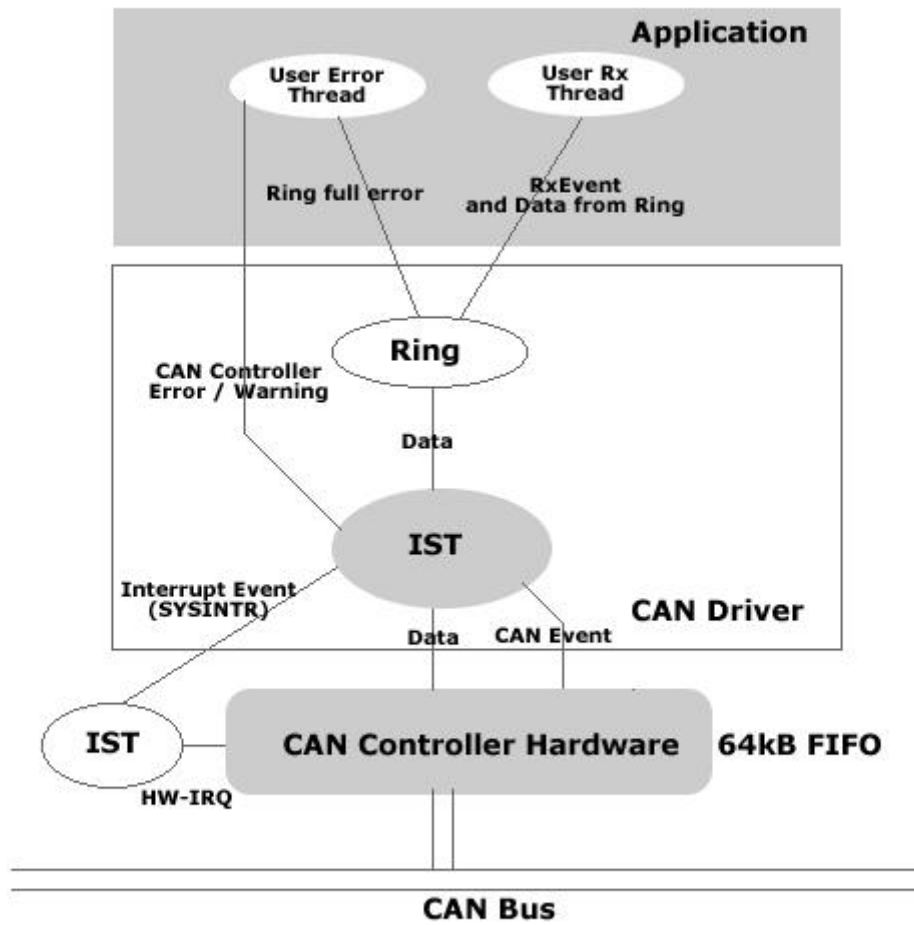
Functions in the CAN driver

```
BOOL  InitializeCANController  (HANDLE hOpenContext);
BOOL  DeinitializeCANController (HANDLE hOpenContext);
BOOL  SWResetCANController     (HANDLE hOpenContext, BOOL bOnOff);
BOOL  SendFrame                (HANDLE hOpenContext, MessageFrame *Msg);
DWORD GetErrorCode             (HANDLE hOpenContext);
void  SetBaudrate              (HANDLE hOpenContext, BYTE index);
#ifdef HCAN2
void  SetGlobalAcceptanceFilter (HANDLE hOpenContext, BYTE *Filter, BYTE size);
#endif
DWORD GetCANMemorySnapShot(HANDLE hOpenContext, BYTE *ptr, DWORD size);
```

CANBase_SJA1000.cpp / I82527.cpp / HCAN2.cpp

Hardware-Dependent functions for CAN controller access and interrupt handling.

Data Flow und Events



CAN Driver Interface CAN_API.H

```
BOOL CAN_StartChip (HANDLE hDevice);
BOOL CAN_StopChip (HANDLE hDevice);
BOOL CAN_GetNextReceivedFrame(HANDLE hDevice, MessageFrame *RxFrameBuffer);
BOOL CAN_SendFrame (HANDLE hDevice, MessageFrame *TxFrameBuffer);
BOOL CAN_GetError (HANDLE hDevice, DWORD *ErrorPtr);
BOOL CAN_SetBaudRate(HANDLE hDevice, BYTE *index);

#define SINGLE_FILTER_MODE 0x01
#define DUAL_FILTER_MODE 0x00
BOOL CAN_SetAcceptanceFilter(HANDLE hDevice, ACCEPTFILTER *AcceptanceFilter);
```

CAN_API.H contains easy-to-use functions to access the CAN stream driver from a user mode application. The functions in CAN_API.CPP use DeviceIoControl() to communicate with the CAN_DRIVER.DLL.

```
BOOL CAN_StartChip (HANDLE hDevice); // IOCTL_START_CHIP
```

CAN_StartChip in STREAM.CPP calls the lower-level function SWResetCANController(0). Calling this function with argument 0 will put the CAN Controller into Operating Mode.

```
BOOL CAN_StopChip (HANDLE hDevice); // IOCTL_STOP_CHIP
```

CAN_StopChip in STREAM.CPP calls the lower-level function SWResetCANController(1). Calling this function with argument 1 will put the CAN Controller into Software Reset Mode.

Hardware dependent definition of a MessageFrame with Standard or Extended Identifier (in this case SJA1000):

```
typedef union {
    struct
    {
        BYTE FrameInfo; // CAN address 16
        BYTE ID[2];     // CAN address 17-18
        BYTE Data[8];   // CAN address 19-26
    } SFF;
    struct
    {
        BYTE FrameInfo; // CAN address 16
        BYTE ID[4];     // CAN address 17-20
        BYTE Data[8];   // CAN address 21-28
    } EFF;
} MessageFrame;
```

```
BOOL CAN_GetNextReceivedFrame(HANDLE hDevice, MessageFrame *RxFrameBuffer);
// IOCTL_RECEIVE_MESSAGE
```

CAN_GetNextReceivedFrame in STREAM.CPP calls the lower-level function RxRing_GetMessage. It returns the next CAN message from the ring buffer, if one is available.

```
BOOL CAN_SendFrame (HANDLE hDevice, MessageFrame *TxFrameBuffer); // IOCTL_SEND_MESSAGE
```

CAN_SendFrame in STREAM.CPP calls the lower-level function SendFrame. It sends the message to the CAN bus. If a previous message has not been completely sent (transmit pending), the function will wait until the transmission is complete.

```
BOOL CAN_GetError (HANDLE hDevice, CAN_ERROR *ErrorPtr); // IOCTL_GET_ERROR
```

CAN_GetError in STREAM.CPP calls the lower-level function CAN_GetErrorState. This function fills the CAN_ERROR structure with possible failures in the driver or in the CAN controller.

```
BOOL CAN_SetBaudRate(HANDLE hDevice, BYTE index); // IOCTL_SET_BAUDRATE
```

CAN_SetBaudRate in STREAM.CPP calls the lower-level function SetBaudrate. This function allows to set the baudrate parameters easily through a timing table with preconfigured values.

Sample User Code:

```
BOOL UserRxThread(HANDLE hDevice)
{
    MessageFrame RxFrameBuffer;

    // receive messages in a loop
    do
    {
        WaitForSingleObject( CanUserEvents.hCANRxEvent, INFINITE );

        while(CAN_GetNextReceivedFrame(hDevice, &RxFrameBuffer)==TRUE)
        {
            g_RxMsg_Counter++;

            if(g_LogFileMode)
                StoreCANPacketToLog(&RxFrameBuffer);

            if(g_EchoMode)
            {
                // send the same packet (echo)
                if(CAN_SendFrame(hDevice, &RxFrameBuffer)==FALSE)
                    MessageBox (NULL, L"CAN_SendFrame returned failure" ,
                                L"CAN Bus API" , MB_ICONERROR|MB_OK);
                g_TxMsg_Counter++;
            }
        }
    }while(!TerminateUserRxThread);

    return TRUE;
}
```

3. Registry Configuration

The bus interface must be configured according your hardware. The value depends on the can interface to the bus driver.

```
; SJA1000 Output Control Register (Bus Interface), Default: 0xDA
; Default Bus Interface Configuration
;   Output Control: DA
;   normal output mode
;   TX0 Push-pull
;   TX0 Output Polarity 0
;   TX1 Push-pull
;   TX1 Output Polarity 0
"OutputControl" = dword:0DA
```

Windows CE Interrupt Number SYSINTR_

```
; use hardware irq 5 (GIISR.DLL will be used)
"IRQ" = dword:5
; if SysIntr is 0 the value will be automatically generated
; by the driver (RequestSysIntr)
"SysIntr" = dword:0
```

Interrupt Service Thread Priority

0 through 96	Reserved for real-time above drivers.
97 through 152	Used by the default Windows CE-based device drivers.
153 through 247	Reserved for real-time below drivers.
248 through 255	Maps to non-real-time priorities.

```
"IST_ThreadPrio" = dword:10
```

Sample configuration for a memory mapped can controller (x86): The controller is memory mapped at physical address D000:0h

```
"BaseAddress" = dword:800D0000
"MemMapped" = dword:1
"MemSize" = dword:200
"MapPagePhysical" = dword:0
```

The following key enables support for the CAN card CPCXT from EMS:

```
"EMSCard" = dword:1
```

Note that this card uses 512 Bytes of memory (200h).

Sample configuration for a I/O mapped CAN controller:

```
"BaseAddress" = dword:300
"MemMapped" = dword:0
```

Sample configuration for a single ISA can card (EMS Wünsche CPC_XT):

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN1]
  "Prefix"="CAN"
  "Dll"="can_driver.dll"
  "Order"=dword:0
  "Index"=dword:1

; SJA1000 Output Control Register (Bus Interface), Default: 0xDA
"OutputControl" = dword:0DA

; use hardware irq 5
"Irq" = dword:5
; if SysIntr is 0 the value will be automatically generated by the driver (RequestSysIntr)
"SysIntr" = dword:0
"IsrDll"="giisr.dll"
"IsrHandler"="ISRHandler"
"IST_ThreadPrio" = dword:10

; sample configuration for memory mapped can controller
"MemBase" = dword:800D0000
"MemMapped" = dword:1
"MemLen" = dword:200
"MapPagePhysical"= dword:0
"EMSCard" = dword:1

; sample configuration for I/O mapped can controller
; "BaseAddress" = dword:300
; "MemMapped" = dword:0
```

Sample configuration for multiple CAN controllers

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN1]
  "Prefix"="CAN"
  "Dll"="can_driver.dll"
  "Order"=dword:0
  "Index"=dword:1

  ; SJA1000 Output Control Register (Bus Interface), Default: 0xDA
  "OutputControl" = dword:0DA

  ; use hardware irq 5
  "IRQ" = dword:5
  ; if SysIntr is 0 the value will be automatically generated by the driver
(RequestSysIntr)
  "SysIntr" = dword:0
  "IST_ThreadPrio" = dword:10

  ; sample configuration for memory mapped can controller
  "BaseAddress" = dword:800D0000
  "MemMapped" = dword:1
  "MemSize" = dword:100
  "MapPagePhysical"= dword:0
  "EMSCard" = dword:0

  ; sample configuration for I/O mapped can controller
  ; "BaseAddress" = dword:300
  ; "MemMapped" = dword:0

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN2]
  "Prefix"="CAN"
  "Dll"="can_driver.dll"
  "Order"=dword:1
  "Index"=dword:2

  ; SJA1000 Output Control Register (Bus Interface), Default: 0xDA
  "OutputControl" = dword:0DA

  ; use hardware irq 5
  "IRQ" = dword:5
  ; if SysIntr is 0 the value will be automatically generated by the driver
(RequestSysIntr)
  "SysIntr" = dword:0
  "IST_ThreadPrio" = dword:10

  ; sample configuration for memory mapped can controller
  "BaseAddress" = dword:800D0200
  "MemMapped" = dword:1
  "MemSize" = dword:100
  "MapPagePhysical"= dword:0
  "EMSCard" = dword:0

  ; sample configuration for I/O mapped can controller
  ; "BaseAddress" = dword:300
  ; "MemMapped" = dword:0
```

Use "can_api_test COM2:" to work with the second CAN controller instance.

Sample configuration for the EMS PCI card CPC_PCI with 2 can controllers:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PCI\Template\PCICAN]
  "Prefix"="EMS"
  "Dll"="can_driver.dll"
  "Order"=dword:0
  "Class"=dword:02
  "SubClass"=dword:80
  "ProgIF"=dword:00
  "VendorID"=multi_sz:"110A"
  "DeviceID"=multi_sz:"2104"
  "IsrDll"="giisr.dll"
  "IsrHandler"="ISRHandler"

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PCI\Template\PCICAN\Device0]
  "Prefix"="CAN"
  "Dll"="can_driver.dll"
  "Order"=dword:0
  "Index"=dword:1

  ; SJA1000 Output Control Register (Bus Interface), Default: 0xDA
  "OutputControl" = dword:0DA

  "IST_ThreadPrio" = dword:10

  ; sample configuration for memory mapped can controller
  "MemMapped" = dword:1
  "MapPagePhysical"= dword:1
  "EMSCard" = dword:0

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PCI\Template\PCICAN\Device1]
  "Prefix"="CAN"
  "Dll"="can_driver.dll"
  "Order"=dword:0
  "Index"=dword:2

  ; SJA1000 Output Control Register (Bus Interface), Default: 0xDA
  "OutputControl" = dword:0DA

  "IST_ThreadPrio" = dword:10

  ; sample configuration for memory mapped can controller
  "MemMapped" = dword:1
  "MapPagePhysical"= dword:1
  "EMSCard" = dword:0
```

Sample configuration for Hitachi HCAN2:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN1]
"Prefix"="CAN"
"Dll"="can_driver.dll"
"Order"=dword:0
"Index"=dword:1

; SysIntr must be the same in file OALINTR.H
; #define SYSINTR_HCAN0          (SYSINTR_FIRMWARE+1)
"SysIntr"          = dword:11
"IST_ThreadPrio"  = dword:10

; sample configuration for memory mapped can controller
"MemBase"         = dword:FE380000
"MemMapped"       = dword:1
"MemLen"          = dword:500
"MapPagePhysical"= dword:1

[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\CAN2]
"Prefix"="CAN"
"Dll"="can_driver.dll"
"Order"=dword:1
"Index"=dword:2

; SysIntr must be the same in file OALINTR.H
; #define SYSINTR_HCAN1          (SYSINTR_FIRMWARE+23)
"SysIntr"          = dword:27
"IST_ThreadPrio"  = dword:10

; sample configuration for memory mapped can controller
"MemBase"         = dword:FE390000
"MemMapped"       = dword:1
"MemLen"          = dword:500
"MapPagePhysical"= dword:1
```