CAN Treiber lmscanrv OS9 v1.0
EMUF PPC 555

Version 1.0: feste Baudrate 500 kBaud, 11-Bit-Identifier auch "Base frame format" genannt (CAN 2.0A)

Der CAN-Treiber lmscandrv besteht aus folgenden Komponenten:

Devicedescriptor	dlmscana	CAN A Anschluss am EMUF
Devicedescriptor	dlmscanb	CAN B Anschluss am EMUF
Driver	lmscandrv	
Headerdatei	lmscandrv_client.h	include Datei für Anwendungen

Zu dieser Doku gehört das File *lmscandrv_18052008.zip* in den die o.g. Dateien enthalten sind.

Der Treiber *lmscandrv* nutzt den SCF Filemanager von OS-9. Die Devicedesciptoren sind für den SCF so eingestellt, dass alle Zeichen 1:1 binär 8-bittig ohne Echo durchgereicht werden; sowohl in Sende- als auch in Empfangsrichtung. Der Treiber arbeitet im sog. Interruptbetrieb.

Es wird vom und zum Treiber immer ein CAN-Telegrammrahmen übertragen. Der Aufbau dieses Rahmens ist in der Struktur *CANTELE* in *lmscandrv_client.h* dokumentiert. Pro Schreib- oder Lesevorgang wird immer in ganzzahligen Vielfachen von *sizeof(CANTELE)* Bytes übertragen.

CANTELE-Zusammensetzung erklärt anhand lmscandrv_client.h:

```
#ifndef _LMSCANDRV_CLIENT
#define _LMSCANDRV_CLIENT


#define LMS_CANA_RX "LMS_CANA_RX"
#define LMS_CANB_RX "LMS_CANB_RX"

typedef struct {
        u_int16 xtim; /*on _os_read() timestamp on _os_ write() it must be 0xa55a */
        u_int16 stat; /* not used */
        u_int16 id;  /* CAN ID 11-bit*/
        u_int16 len; /*Anzahl 1-8 der zu sendenden Datenbytes, die folgen*/
        u_int16 rtr; /* not used up to now*/
        u_int8 da[8]; /*8 Datenbytes maximal*/
} CANTELE;

typedef union {
CANTELE c;
u_int8 b[sizeof(CANTELE)];
} W_CAN;
#endif
```

Funktionsweise ist wie bei der Kommunikation mit einer seriellen Schnittstelle oder Datei (für Fehlercodes und Standardverhalten der unten aufgeführten *_os_xx()*-Funktionen siehe UltraC Lib-Reference Manual):

#include <modes.h> *#include <types.h>* *#include <sg_codes.h>* *#include <const.h>*			**Includes für alle Funktionen**
error_code _os_open(char canpath, u_int32 mode, path_id *path);*	*input* *input* *output*	*canpath=* *"/dlmscana"* *"/dlmscanb"* *mode=* *S_IREAD* *S_IWRITE* *path*	**Öffnet den CAN A/B Kanal** (und es wird ein Event angelegt siehe unten) lesen schreiben CAN Kanal Handle für Lese- Schreibaufrufe
error_code _os_close(path_id path);	*input*	*path*	**Schliesst den CAN Kanal *path*** *und entfernt das event*
*error_code _os_gs_ready(path_id path, u_int32 *incount);*	 *input* *output*	 *path* *incount*	**Anzahl der abzuholenden CAN Telgramme** CAN Kanal Handle Anzahl der Bytes zum Abholen; die Anzahl der noch nicht gelesenen CAN-Telegramme berechnet sich zu: *(int)(incount/sizeof(CANTELE))*
*error_code _os_read(path_id path,void *buffer,u_int32 *count);*	 *input* *input* *input*	 *path* *buffer* *count*	**Blockierendes Lesen** CAN Kanal Handle Adresse CAN-telgramm-Empfangsbuffers count= can_anz*sizeof(CANTELE) can_anz ist die Anzahl der abzuholenden CAN-Telegramme, count die Anzahl der Bytes!
*error_code _os_write(path_id path,void *buffer,u_int32*count);*	 *input* *input* *input*	 *path* *buffer* *count*	**Blockierendes Schreiben** CAN Kanal Handle Adresse des ersten zu sendenden CAN-telgramms count= can_anz*sizeof(CANTELE) can_anz ist die Anzahl der zu senden CAN-Telegramme, count die Anzahl der Bytes!
Beim Öffnen des CAN A/B Kanals vom Treiber angelegtes Event:			
LMS_CANA_RX		*init value=0* *sinc=1* *winc=-1*	Mit jedem CAN-Telegramm-Empfang auf CAN A Kanal wird das Event um eins erhöht
LMS_CANB_RX		*init value=0* *sinc=1* *winc=-1*	Mit jedem CAN-Telegramm-Empfang auf CAN B Kanal wird das Event um eins erhöht

Achtung: Beim CAN-Telegramm senden *_os_write()*-Aufruf muss sichergestellt sein, dass im übergebenen Sendeobjekt des Typs *CANTELE* die Stukturkomponente *xtim* mit *0xa55a* belegt ist.

Der CAN-Treiber *lmscandrv* wird entweder entweder interaktiv via iniz /dlmscana bzw. *iniz /dlmscanb* initalisiert oder durch ein *_os_open()*-Aufruf in der Applikation. Der entsprechende *_os_close()* -Aufruf deinitialisiert, wenn kein weiterer das Gerät geöffnet hat, den Treiber wieder. Interaktiv ist dies durch das Kommando deiniz /dlmscana bzw. *iniz /dlmscanb* möglich.

Typischer CAN-Telgramm Lese-Ablauf mit Eventauswertung:

CANTELE cantele; path_id pwd; event_id arx;

1. *Öffenen des CAN B-Kanals zum lesen und schreiben:* ***_os_open("/dlmscanb",S_IREAD | S_IWRITE,&pdw);***
2. *Anlinken auf das Leseevent des CAN B Kanals:* ***_os_ev_link(„LMS_CANB_RX“,&arx);***
3. *Warten auf ein CAN-Telegramm:* ***_os_ev_wait(arx,&val,&sig,1, 0x7FFFFFFEl);***
4. *Lesen eines CAN-Telegramms:* ***cnt=sizeof(CANTELE); _os_read(pdw,&cantele,&cnt);***

Wenn mit Eventauswertung gearbeitet werden soll, so ist ein Abholen eines Telegramms via *_os_read()*-Aufruf ohne vorherigen *_os_ev_wait()*-Aufruf problematisch. Dieser so getätigte *_os_read()*-Aufruf reduziert nicht den Eventvalue um eins. Damit ist der Eventvalue und die Anzahl der tatsächlich vorhandenen abzuholenden CAN-Telegramme nicht mehr synchron. Der Eventvalue zeigt dann mehr Telegramme an als tatsächlich da sind!

Typische Endesequenz einer Applikation mit Eventauswertung:

path_id pwd; event_id arx;

1. *Schiessen Zugriff auf Event* ***_os_ev_unlink(arx);:***
2. *Schliessen des CAN B-Kanals:* ***_os_close(pdw);***

Typischer CAN-Telgramm Schreib-Ablauf:

CANTELE cantele; path_id pwd;

1. *Öffenen des CAN B-Kanals zum lesen und schreiben:* ***_os_open("/dlmscanb",S_IREAD | S_IWRITE,&pdw);***
2. *Schreiben eines CAN-Telegramms:* ***cnt=sizeof(CANTELE); _os_write(pdw,&cantele,&cnt);***

Typischer CAN-Telgramm Lese Ablauf mit _os_gs_ready()-Verwendung:

CANTELE cantele; path_id pwd;

1. *Öffenen des CAN B-Kanals zum lesen und schreiben:* ***_os_open("/dlmscanb",S_IREAD | S_IWRITE,&pdw);***
2. *Testen ob ein CAN-Telgramm da ist:* ***do { _os_gs_ready(pwd,&cnt);} while ((cnt/sizeof(CANTELE))==0);***
3. *Lesen eines CAN-Telegramms:* ***cnt=sizeof(CANTELE); _os_read(pdw,&cantele,&cnt);***

Typische Endesequenz einer Applikation mit _os_gs_ready()-Verwendung:

path_id pwd;

1. *Schliessen des CAN B-Kanals:* ***_os_close(pdw);***