

Laborversuch 3: CAN/AD Converter

3.1. Ziele des Laborversuchs

- ➔ CAN ist ein asynchrones, bit-serielles Bussystem, das weltweit branchenübergreifend Verwendung in verschiedenen Bereichen finden. Über CAN erfolgt eine sichere und zuverlässige Datenkommunikation für zahlreiche Anwendungen in der Automobil-, in der Verkehrs-, in der Medizintechnik und mit bestimmten Erweiterungen auch in der Automatisierungstechnik. In jedem modernen Kraftfahrzeug tauschen verschiedene elektronische Steuergeräte Daten wie die intern errechnete Geschwindigkeit und die Motoröltemperatur über einen CAN Bus aus. In der Medizintechnik haben sich viele Hersteller von großen Geräten wie beispielsweise von Computertomografen (CT), Magnetresonatoren (MR) und anderen Diagnosegeräten für den CAN Bus entschieden, um Subsysteme (z.B. C-Bogen, Patiententisch, Kollimator, Dosismesssystem) zu vernetzen. Typischerweise läuft der Datenaustausch über das in Software implementiertes, übergeordnetes Protokoll CANopen. In diesem Laborversuch werden Sie lernen, wie Daten über einen CAN Bus gesendet und empfangen werden.
- ➔ Weiterhin stehen die Konfiguration und Verwendung eines AD-Wandlers im Fokus dieses Laborversuchs. Mit einem AD-Wandler kann ein analoges Signal abgetastet und digitalisiert werden. Als Beispiel für die Verwendung eines AD-Wandlers lässt sich die Einstellung der Lautstärke eines digitalen Audio-Geräts (z.B. DVD-Player, MP3-Player, ...) anführen. In diesem Fall misst der AD Wandler die an einem Potentiometer anliegende analoge Spannung. Der erfasste digitale Wert wird in einen Mikrocontroller oder einen digitalen Signalprozessor (DSP) eingespeist, um eine Änderung der Lautstärke mittels eines Software-Programms zu ermöglichen. Die Ansteuerung eines AD-Wandlers mit einem Mikrocontroller steht auch im Fokus dieses Laborversuchs.

3.2. Vorbemerkungen zur Vorbereitung und Durchführung des Laborversuchs

In diesem Laborversuch werden folgende Schnittstellen des LandTiger LPC1768 EVB verwendet:

- ❖ CAN1 und CAN2 Anschlüsse
- ❖ LCD Touch Display Modul
- ❖ Potentiometer (am Pin AD0.5 angeschlossen)

3.3. Vorbereitung

➔ Übung L3-1: Analogwert (AD-Wert) ermitteln

(Sim)

Der Mikrocontroller NXP LPC1768 beinhaltet einen 8-kanaligen AD-Wandler. Am AD Kanal 5 (also am Pin AD0.5) ist ein Potentiometer angeschlossen. Mit dem Potentiometer lässt sich die am Pin AD0.5 anliegende analoge Spannung ändern. In dieser Übung soll der AD-Wandler vom Mikrocontroller NXP LPC1768 in Betrieb genommen werden.

Für den Software-Modus ist nur eine Auflösung von 12 Bits vorgesehen. Die AD-Umsetzung kann maximal mit einer Frequenz von 13 MHz erfolgen. Schreiben Sie eine C-Funktion mit dem Namen **ADC_Config**, um den AD-Wandler AD0 vom Mikrocontroller NXP LPC1768 zu konfigurieren. Die Funktion benötigt keinen Parameter und liefert keinen Wert zurück. Prüfen Sie bitte, ob der AD-Wandler nach einem Reset des Mikrocontrollers automatisch freigeschaltet oder ausgeschaltet ist. Diese Information erhalten Sie im Register PCONP (siehe Reference Manual Chapter 4: LPC17XX clocking and power control).

Schreiben Sie eine Funktion mit dem Namen **ADC_StartConversion**, die zur Durchführung einer AD-Umsetzung dienen soll. Eine neue AD-Umsetzung darf erfolgen, nur wenn eine vorherige AD-Umsetzung bereits abgeschlossen ist. Für diese Funktion werden weder Parameter noch Rückgabewert benötigt.

Schreiben Sie nun ein C-Programm, das eine AD-Umsetzung in Software-Modus **alle 10 mSek** ausführt. Dabei sollen die Funktionen **ADC_Config** und **ADC_StartConversion** aufgerufen werden.

Im Simulator können Sie die Datei **AD0_AnalogSignal (50mV 1ms).ini** verwendet. Nach einem Klick auf den Button **AD0_AnalogSignal** wird ein analoges dreieckiges Signal mit einer Periodendauer von 132ms erzeugt. Mithilfe von Breakpoints und der Ansicht im **Logic Analyzer** Fenster können prüfen, ob die AD-Umsetzung korrekte Werte liefert.

➔ Übung L3-2: Verhalten des CAN Akzeptanzfilters

Für das CAN stehen folgende API-Funktionen (in der Datei **can_api.c**) zur Verfügung:

- ❖ **CAN_Init(uint32_t CAN_Module_Number, uint32_t CAN_Baud_Rate)**
Mit dem Parameter **CAN_Module_Number** wird das zu konfigurierende CAN Modul ausgewählt (**CAN_Module_Number** = 1 für das Modul CAN1 und 2 für CAN2). Der Parameter **CAN_Baud_Rate** entspricht dem Inhalt vom CANxBTR Register (x = 1 oder 2). Diese Funktion gibt den logischen Wert TRUE zurück, wenn sie erfolgreich zum Abschluss gekommen. In der Funktion **CAN_Init** wird auch der Receive Interrupt freigeschaltet.
- ❖ **CAN_Receive_Message(uint32_t CAN_Module_Number, CAN_MSG*Dest_Rx_Data);**
Mit dieser Funktion werden Daten aus den Empfangspuffern CANxRDA und CANxRDB (x = CAN_Module_Number = 1 oder 2) in eine Datenstruktur vom Typ CAN_MSG kopiert. Der Parameter **Dest_Rx_Data** stellt einen Zeiger auf diese Datenstruktur dar. Diese Funktion wird in der Unterbrechungsroutine **CAN_Interrupt_Handler()** aufgerufen.
- ❖ **CAN_Send_Message(uint32_t CAN_Module_Number, CAN_MSG *Tx_Data)**
Der Inhalt eines zu sendenden Datentelegramms wird in einer Datenstruktur vom Typ CAN_MSG vorbereitet. In der Regel wird diese Funktion nicht vom CAN Interrupt-Handler aufgerufen. Ein Aufruf dieser Funktion aus dem Interrupt-Handler ist durchaus möglich. Dafür müsste mindestens ein Transmit Interrupt Bit TIE1, TIE2 oder TIE3 (jeweils für den CAN Senderpuffer 1, 2 oder 3) gesetzt wird.
- ❖ **CAN_Interrupt_Handler(void);**
Die vorgegebene Version vom CAN Interrupt-Handler dient nur zum Empfangen von CAN Datentelegrammen. Empfangene Datentelegramme werden der Reihe nach in eine Liste eingefügt. Sie sollen empfangene Datentelegramme aus dieser vorhandenen Liste entnehmen. Hierfür sollen Sie zuerst prüfen, ob die Variable **CAN1_Counter_RX** oder **CAN2_Counter_RX** größer als 0 ist. Diese Variable zeigt an, wie viele Botschaften über den CAN1 oder CAN2 Controller empfangen wurden.
- ❖ **CAN_Set_Acceptance_Filter_Mode(uint32_t ACF_Mode)**
Mit dieser Funktion wird die Akzeptanzfilterung konfiguriert. Der Wert vom Parameter ACF_Mode legt fest, ob ankommende Datentelegramme vollständig oder selektiv nach (einer Ausfilterung) zugelassen oder ignoriert werden:
 - > ACF_Mode = 1 (=ACCF_OFF) → Kein Datentelegramm wird zugelassen.
 - > ACF_Mode = 2 (=ACCF_BYPASS) → Alle Datentelegramme werden zugelassen.
 - > ACF_Mode = 0 (= ACCF_ON) → Nur Datentelegramme mit vorgegebenen ID Werten werden zugelassen. Der Empfang von Datenlegrammen erfolgt mit den Registern CANxRDA und CANxRDB. Die zuzulassenden ID Werte werden in der Datei **CAN_AcceptanceFiltering.h** vorgegeben.

- > ACF_Mode = 4 (=ACCF_FULLCAN) → Der Empfang von Datentelegrammen erfolgt über einen internen Speicher CAN LUT (Lookup Table) RAM.

❖ CAN_Set_Acceptance_LUT_RAM(void)

Diese Funktion dient zur Einstellung der ID Werte von CAN Datentelegrammen, die über einen internen Speicher CAN LUT (Lookup Table) RAM empfangen werden.

In der Datei **CAN_AcceptanceFiltering.h** können Sie die zuzulassenden ID Werte vorgeben, wenn die Option ACF_Mode = ACCF_ON = 0 ausgewählt wird. In diesem Labor werden wir diese Option verwenden, wenn CAN Datentelegramme selektiv empfangen werden sollen.

Weiterhin werden CAN Datenlegramme nach dem Standard CAN 2.0A Format (d.h. mit 11 ID Bits) übertragen. In der Datei **CAN_AcceptanceFiltering.h** werden nur die folgenden Zeilen angepasst, um ein selektives Empfangen von Datentelegrammen zu ermöglichen:

```
const uint32_t  CAN1_Number_SFF_EXP_ID = 2;
const uint16_t  CAN1_Array_SFF_EXP_IDs[CAN1_Number_SFF_EXP_ID] = {0x100, 0x456};
```

und/oder

```
const uint32_t  CAN2_Number_SFF_EXP_ID = 2;
const uint16_t  CAN2_Array_SFF_EXP_IDs[CAN1_Number_SFF_EXP_ID] = {0x100, 0x456};
```

SFF_EXP_ID steht für Standard Frame Format / Explicit ID.

Es wird zuerst die Anzahl der zuzulassenden ID Werte angegeben. Danach werden die zuzulassenden ID Werte explizit aufgelistet. In diesem Fall werden CAN Datentelegramme mit dem ID Wert 0x100 oder 0x456 zugelassen. Ein Receive Interrupt wird nur für CAN Datentelegramme mit dem ID Wert 0x100 oder 0x456 ausgelöst.

In dieser Übung soll der Empfang von CAN Datentelegrammen getestet werden. Im Verzeichnis L3_Uebung2 befinden sich folgende Dateien: **CAN_AcceptanceFiltering.h**, **can_api.c**, **can_test_rx.c**, **emb1_can_labs.h** und **L3_Uebung2.ini**.

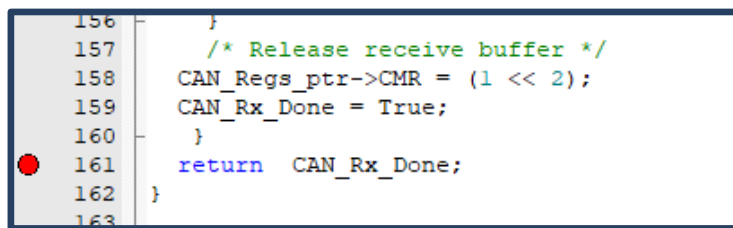
Mit der ini Datei **L3_Uebung2.ini** werden CAN Datentelegramme gesendet werden, wenn man auf den Button **CAN Send Msg** klickt. Hierbei weist das erste Datentelegramm den ID Wert 0xFC auf. Daraufhin werden weitere CAN Datentelegramme jeweils mit einem um +1 inkrementierten ID Wert übertragen.

Passen Sie die ID Werte in der Header Datei **CAN_AcceptanceFiltering.h** an, um aus der Reihe der zu sendenden ID-Werte auswählen zu können, welche CAN Datentelegramme zugelassen oder ignoriert werden sollen.

Die Schablonendatei **can_test_rx.c** soll vervollständigt werden, so dass das zu verwendende CAN Modul (CAN1 oder CAN2) initialisiert wird. Nach der Initialisierung soll eine Einstellung der Akzeptanzfilterung erfolgen. Danach soll eine endlose Schleife vorgesehen werden, die immer unterbrochen wird, wenn ein zugelassenes CAN Datentelegramm einen Interrupt auslöst.

Prüfen Sie bitte auch die Filterungsmodi ACCF_OFF, ACCF_BYPASS und ACCF_ON.

Setzen einen Break Point am Ende der Funktion *CAN_Receive_Message (void)*, um festzustellen, eine CAN Botschaft tatsächlich empfangen wurde.



```

156     }
157     /* Release receive buffer */
158     CAN_Regs_ptr->CMR = (1 << 2);
159     CAN_Rx_Done = True;
160 }
161 return CAN_Rx_Done;
162 }
163

```

➔ Übung L3-3: Loopback-Funktion

Nun soll die CAN Sendefunktion mithilfe einer Verbindung zwischen den 2 CAN Controllern CAN1 und CAN2 geprüft werden. Ein CAN Datentelegramm wird in Form einer Datenstruktur vorbereitet, indem konkrete Werte für die verschiedenen Felder im C Code angegeben werden. Nach einer Warteschleife soll ein CAN Datentelegramm gesendet werden. Hierbei soll die Warteschleife wie folgt implementiert werden:

```
for (i=0; i < 100000; i++);
```

Denken Sie auch dran, die CAN Module vorher zu initialisieren. Im Verzeichnis **L3_Uebung3** finden Sie die benötigten Vorlagendateien. Mit der ini Datei **CAN_LoopBack.ini** lässt sich eine Verbindung zwischen den CAN Controllern realisieren:

- Button **LoopBack_1to2** → CAN1 sendet und CAN2 empfängt
- Button **LoopBack_2to1** → CAN2 sendet und CAN1 empfängt

Sie können die Schablonendatei **can_rx_tx.c** verwenden, um zu üben, wie Daten mit einem CAN Controller gesendet und empfangen werden. Denken Sie bitte daran, ein zu sendendes Datentelegramm vorher vorzubereiten (d.h., ID Wert, DLC Wert und Payload Bytes).

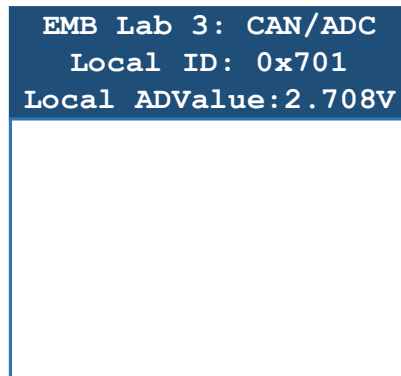
Die zuletzt empfangenen Daten befinden sich in der Variable **My_CAN1_RX_message** oder **My_CAN2_RX_message**. Prüfen Sie die verschiedenen Modi der Akzeptanzfilterung: ACCF_OFF, ACCF_BYPASS und ACCF_ON.

Im ACCF_ON Modus sollen die durchzulassenden ID Werte in der Datei **CAN_Acceptancefiltering.h** je nach Bedarf angepasst werden. Z.B. Im Falle von Loopback_1_to_2 sendet der CAN1 Controller mit dem ID Wert 0x701. Im ACCF_ON Modus soll der CAN2 Controller diesen ID Wert durchlassen.

3.4. Versuchsdurchführung

3.4.1. Prolog L3-4: AD-Wert als Spannung auf dem LCD

Portieren Sie die Übung L3_Übung1 auf die Ziel-Hardware. Für die Übung L3_Übung1 soll der erfasste **AD Wert in Volt mit 3 Stellen nach dem Komma** berechnet werden und wie folgt auf der LCD Anzeige ausgegeben werden:



Es wird hier angezeigt, dass eine analoge Spannung von 2,708 V lokal erfasst wurde. Man erinnere sich daran, dass ein Spannungswert von 3,3 V dem AD Wert 0xFFF entspricht.

Der Text wird in der Vorlage zur Verfügung gestellt. Es soll nur der analoge Wert hinzugefügt werden. Hierbei können Sie die Funktion

void GLCD_DisplayString (unsigned int ln, unsigned int col, unsigned char *s)

verwenden. Die Parameter **ln** und **col** dienen jeweils zur Auswahl der Zeile und der Spalte auf dem Display. Mit der eingestellten Größe von Zeichen können 10 Zeilen à 20 Zeichen angezeigt werden.

Ihre Aufgabe besteht darin, den C-Code aus der Vorlage zu ergänzen, so dass der am AD Wandler erfasste Spannungswert ständig auf dem LCD Display angezeigt wird. Wie im obenstehenden Bild dargestellt, soll der analoge Spannungswert immer an der gleichen Stelle auf dem LCD Display angezeigt werden.

3.4.2. Aufgabe L3-5: Spannungswert lokal und remote über CAN anzeigen

Die analoge Spannung am Potentiometer soll mit einer Abtastfrequenz von 4 Hz umgesetzt werden. Nach jeder AD Umsetzung wird der erfasste Spannungswert auf der LCD Anzeige ausgegeben. Falls eine Veränderung von mehr als 0,5 V festgestellt wird, soll der neue Spannungswert über den CAN Bus gesendet werden. Hierbei ist **die Differenz zwischen dem zuletzt über den CAN Bus gesendeten Spannungswert und dem neuen gemessenen Spannungswert gemeint**. Der ID Wert dieses CAN Telegramms ergibt sich aus der Summe von der **hexadezimalen** Zahl 0x700 und der Nummer des Laborplatzes.

Beispiel: Vom Laborplatz 10 werden CAN Datentelegramme mit dem ID Wert 0x70A gesendet.

Der Spannungswert soll als eine Zeichenkette (d.h. als eine Folge von ASCII Zeichen) über den CAN Bus gesendet werden, da nur Zeichenketten auf der LCD Anzeige veranschaulicht werden können. Hierbei kann die Funktion **sprintf** verwendet werden.

Beispiel: Für eine analoge Spannung von 1,917 V zeigt der AD Wandler den Wert 0x253. Im C Code kann sich der analoge Wert wie folgt in eine Zeichenkette umgewandelt werden.

```
sprintf(AD_Val_string, "%.3f", AD_AnalogValue);
```

mit den folgenden Deklarationen der entsprechenden Variablen

```
char AD_Val_string[7];
float AD_Voltage;
```

Für den vorgegebenen Wert der analogen Spannung werden folgende Zeichen in einem CAN Datentelegramm übertragen: 0x31, 0x2E, 0x39, 0x31 und 0x37.

Der von **einer** anderen Gruppe gemessene Spannungswert soll auch angezeigt werden. Hierfür soll zuerst die Akzeptanzfilterung aktiviert werden, indem die Funktion **CAN_Set_Acceptance_Filter_Mode** mit der Option **ACCF_ON** aufgerufen wird. Der ID Wert, mit dem die andere Gruppe sendet, soll auch in die Liste `CAN1_Array_SFF_EXP_IDs[]` eingetragen werden.

Im Labor wird ein Tool jede Minute ein CAN Telegramm mit ID Wert 0x700 senden. Die in diesem CAN Telegramm gesendeten Nutzdatenbytes („Payload“) entsprechen der Fließkommazahl 1.234 oder 9.999. Die mit dem ID 0x700 empfangenen Nutzdatenbytes können für Testzwecke an die LCD Anzeige weitergeleitet werden. Mit einem geeigneten Breakpoint können Sie prüfen, ob die folgenden Nutzdatenbytes 0x31, 0x2E, 0x32, 0x33 und 0x34 (für 1.234) oder 0x39, 0x2E, 0x39, 0x39 und 0x39 (für 9.999) mit dem ID Wert 0x700 empfangen werden.

Die LCD Anzeige wird dann wie folgt aussehen:

Die Aufgabe besteht nun darin, ID Werte (d.h. Wert nach **Remote ID:** in der Anzeige) und AD Spannungswerte (d.h.: Wert nach **Remote ADVal.:**) kontinuierlich mit den Inhalten empfangener CAN Botschaften zu aktualisieren.

```
EMB Lab 3: CAN/ADC
Local ID: 0x701
Local ADValue: 2.708V
```

```
=Data Received@CAN1=
Remote ID: 0x700
Remote ADVal.: 1.234V
@@@@@@@@@@@@@@@@@@@@
Remote ID: 0x708
Remote ADVal.: 2.088V
```

Der Code aus der Vorlage sorgt dafür, dass ein ID Wert mit dem entsprechenden analogen Spannungswert nur einmal angezeigt wird, während sich der analoge Wert ändern kann. Daher können sich nur Daten von verschiedenen ID Werten gleichzeitig angezeigt werden.

Ihre Aufgabe besteht nun darin, den C-Code aus der Vorlage zu ergänzen, so dass der empfangene ID Wert (Remote ID) sowie der empfangene analoge Spannungswert (Remote ADVal.) fortlaufend auf dem LCD Display angezeigt werden. Wie im obenstehenden Bild dargestellt, sollen der ID Wert und der analoge Spannungswert immer an den zwei vorgesehenen Stellen auf dem LCD Display angezeigt werden. Sie sollen die benötigten Zeichenketten („Strings“) beispielsweise in den Variablen „ID_String“ und „AD_Val_String“ bereitstellen und folglich die entsprechenden GLCD Ausgabefunktionen ergänzen.