Laborversuch 3: CAN/AD Converter

Name	*Vorname*	*Studien-gang*	*Matr.-Nr.*	

Gruppennummer:

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

➔ Die Vorbereitung des Laborversuchs erfolgt anhand eines taktgenauen Simulationsmodells des Mikrocontrollers. Lösungen zu den im Laborversuch gestellten Übungen werden mit dem Simulator in Form von Hausaufgaben entwickelt und getestet.

➔ Die mit dem Simulator entwickelten Lösungen werden von einem Betreuer ausgewertet. **Eine positive Auswertung gilt als notwendige Bedingung für die weitere Durchführung des Laborversuchs**. Während der Durchführung des Laborversuchs werden Lösungen, die mit dem Simulator entwickelt wurden, auf die Ziel-Hardware portiert und anschließend getestet.

➔ Als Hardwareplattform wird das **EVB (Evaluation Board) MCB2300** von der Firma Keil verwendet. Das MCB2300 EVB (siehe **Abbildung 2**) unterstützt ARM-Mikrocontroller der LPC23xx Familie von der Firma NXP (Ehemalige Halbleiter-Sparte von Philips). Für diesen Laborversuch sind die ausgewählten MCB EVBs mit einem LPC2368 oder LPC2378 Mikrocontroller bestückt und bieten zahlreiche Schnittstellen (CAN, UART, Ethernet, USB, …) sowie eine LCD-Anzeige mit 2 Zeilen à 16 Zeichen an.

In diesem Laborversuch werden folgende Schnittstellen des MCB2300 EVB verwendet:

- ❖ CAN1 und CAN2 Anschlüsse
- ❖ LCD-Anzeige
- ❖ Potentiometer (am Pin AD0.0 angeschlossen)

3.3. Vorbereitung

➔ **L3_Übung1**

Der Mikrocontroller NXP LPC2300 beinhaltet einen 6-kanaligen AD-Wandler. Am Kanal 0 (also am Pin AD0.0) ist ein Potentiometer angeschlossen. Mit dem Potentiometer lässt sich die am Pin AD0.0 anliegende analoge Spannung ändern. In dieser Übung soll der AD-Wandler vom Mikrocontroller NXP LPC2300 in Betrieb genommen werden.

Für den Software-Modus ist nur eine Auflösung von 10 Bits vorgesehen. Die AD-Umsetzung kann maximal mit einer Frequenz von 4,5 MHz durchgeführt werden. Schreiben Sie eine C-Funktion mit dem Namen ***ADC_Config***, um den AD-Wandler AD0 vom Mikrocontroller NXP LPC2300 im Softwaremodus 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 Chapter 4: LPC23XX 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 100 mSek ausführt. Dabei sollen die Funktionen ***ADC_Config*** und ***ADC_StartConversion*** aufgerufen werden.

Im Simulator können Sie die Datei **ADC.ini** verwendet. Nach einem Klick auf den Button **AD0_AnalogSignal** wird ein analoges dreieckiges Signal mit einer Periode von ca. 2 Sek erzeugt. Mithilfe von Breakpoints und der Ansicht im **Logic Analyzer** Fenster können prüfen, ob die AD-Umsetzung korrekte Werte liefert.

➔ **L3_Übung2**

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.

- **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 Riehe nach in eine Liste eingefügt.

- **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 = 3 (= 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** kann ein CAN Datentelegramm gesendet werden, wenn man auf den Button **CAN Send Msg** klickt. Ändern Sie den ID Wert in der ini Datei und/oder in der Header Datei **CAN_AcceptanceFiltering.h**, um zu sehen, wie Botschaften zugelassen oder ignoriert werden.

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, 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.

➔ **L3_Übung3**

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 eine Verbindung zwischen den CAN Controllern realisieren:

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