

# **Embedded Computing**

## **Laborübung Aufgabenstellung**

**FH-Prof. DI. Dr. Robert Okorn**  
**DI (FH) Markus Krenn, MSc**

Stand: 29.09.2016

# Inhalt

1.	Crazy Car Platine – Einführung, I/O-Port Konfiguration .....	1
1.1.	Inhalt.....	1
1.2.	Crazy Car Platine .....	1
1.3.	Softwarestruktur .....	2
1.4.	Code Composer Studio .....	4
1.5.	Anlegen eines Basis Projektes .....	7
1.6.	Schnittstellen zwischen Softwaremodulen .....	7
1.7.	I/O Port Konfiguration .....	8
2.	Digitale Ein-/Ausgabe .....	9
2.1.	Inhalt.....	9
2.2.	Rechendauer Performance Test .....	9
2.3.	Allgemeine I/O Interrupt-Steuerung .....	9
2.4.	Interrupts.....	10
3.	Clock System, Timer Konfiguration .....	11
3.1.	Inhalt.....	11
3.2.	Power Management Module .....	11
3.3.	Unified Clock System .....	11
3.4.	Timer Interrupts – Timer B0 .....	12
4.	PWM Erzeugung, Aktorik.....	13
4.1.	Inhalt.....	13
4.2.	Timer A1 – PWM Erzeugung.....	13
4.3.	Driver Layer - Aktorik.....	14
4.4.	Drehzahlmessung (optional) .....	14
5.	SPI .....	15
5.1.	Inhalt.....	15
5.2.	USCI B1 - Konfiguration .....	15
5.3.	USCIA0 - UART Konfiguration (optional) .....	17
6.	SPI .....	18
6.1.	Inhalt.....	18
7.	LC - Display .....	19
7.1.	Inhalt.....	19
7.2.	Funktionen des Displays .....	19
7.3.	Display Driver .....	19
7.4.	Fast Display Driver (optional) .....	21
8.	ADC.....	22
8.1.	Inhalt.....	22
8.2.	ADC Konfiguration .....	22
9.	ADC .....	24
9.1.	Inhalt.....	24
9.2.	ADC - DMA Konfiguration .....	24
10.	Sharp Abstandssensoren .....	25
10.1.	Inhalt.....	25
10.2.	Sensor Kennlinie .....	25
11.	Fahralgorithmen.....	26
11.1.	Allgemein.....	26
11.2.	Sensoren .....	26
11.3.	Aktorik .....	27
11.4.	Regelalgorithmen .....	27
11.5.	Fahralgorithmen .....	28

## Abbildungsverzeichnis

Abbildung 1: Screenshot Project Explorer CCS - Struktur .....	3
Abbildung 2: Screenshot Project Explorer – Software Basis Struktur .....	5
Abbildung 3: Screenshot Code Composer Studio – Include Optionen.....	5
Abbildung 4: Screenshot Code Composer Studio – Register View.....	6
Abbildung 5: Flow Chart Statemachine.....	27

# 1. Crazy Car Platine – Einführung, I/O-Port Konfiguration

## 1.1. Inhalt

### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 12: Digital I/O Module

### Wissensüberprüfung

- Hexadezimale/Binäre/Logische Rechenoperation
- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 12: Digital I/O Module
    - 12.1 Digital I/O Introduction
    - 12.2 Digital I/O Operation: 12.2.1 - 12.2.6

## 1.2. Crazy Car Platine

- MSP430F5335 – MSP430 5 Core
- 3 analoge Eingänge für Abstandssensoren
- 1 Hall Effekt Sensor für Drehzahl/Drehrichtungsmessung Hinterachse
- MPU-9250 9 Axis Motion Sensor: Beschleunigung, Gyroskop, Kompass
- Start/Stop Taste
- Ausgänge für die Ansteuerung des Servos / Fahrtenreglers
- LC-Display
- JTAG Spy-by-wire Interface

### 1.3. Softwarestruktur

Die Software soll als Layermodell implementiert werden. Dabei baut jedes Softwaremodul auf das Vorherige auf. Vorteil dieses Modells ist es, dass man einzelne Schichten austauschen kann, ohne die gesamte Software neu schreiben zu müssen. z.B. der Mikrokontrollertyp ändert sich, bei guter Softwarestruktur muss nur der HAL ausgetauscht werden.

Im Zuge der Laborübungen werden alle relevanten Softwareteile behandelt und programmiert mit denen das Crazy Car autonom fahren bzw. durch den Kurs navigieren kann.

#### ***HAL – Hardware Abstraction Layer***

In diesem Layer werden alle Peripheriemodule des Mikrokontrollers initialisiert. Ebenso wird eine Schnittstelle zur nächsten Softwareschicht bereitgestellt. Diese Schnittstelle kann z.B. eine Funktion mit und ohne Parameter oder globale Variablen sein - nicht mischen. Im HAL wird direkt auf Register/Hardware des Mikrokontrollers zugegriffen.

#### ***DL – Driver Layer***

Im Treiber Layer werden fertige Funktionen/Schnittstellen des HAL benutzt um Low Level Funktionen auszuführen. Im DL wird nicht mehr direkt auf Registernamen zugegriffen. Der Driver Layer wird nicht immer benötigt.

#### ***AL – Application Layer***

Hier läuft die eigentliche Applikationslogik ab.

#### **Bsp.: Lenkung des Crazy Cars**

##### HAL

```
HAL_TimerA1_Init();           // Konfiguriert und initialisiert den Timer A1 im PWM Modus
```

Die Werte die dem TimerA1 übergeben werden müssen hängen von der Konfiguration des Timers ab. Deshalb muss eine Schnittstelle definiert werden, die immer denselben Wertebereich übernimmt.

```
HAL_TimerA1_SetSteer(value)
```

Value hat einen fix definierten Wertebereich von. Z.B. -100 ÷ +100 oder 0 ÷ 200.  
Dabei ist 0 => maximaler Lenkusschlag rechts, 200 maximaler Lenkusschlag links.

##### AL

```
HAL_TimerA1_SetSteer(0);
```

Würde man die Konfiguration des Timers ändern, müsste man nur die Umrechnung in der SetSteer Funktion ändern, da es eine definierte Schnittstelle gibt.

##### Optional

##### DL

```
DL_SetSteering();
```

### Bsp.: Datenübertragung zum Display

#### HAL

```
HAL_USCIB1_Init();           // Konfiguriert und initialisiert die USCIB1 Schnittstelle im SPI Modus
HAL_USCIB1_Transmit( data*,len); // Sendet die Anzahl len an Bytes von data
```

#### DL

```
DL_Display_Init();           // Sendet mittels HAL_USCIB1_Transmit Daten an das Display
DL_Display_WriteText();      // Sendet mittels HAL_USCIB1_Transmit Daten an das Display
```

#### AL

```
If(Start_Taste)
    DL_Display_WriteText(„Hello“,5);
If(Stop_Taste)
    DL_Display_WriteText(„World“,5);
```

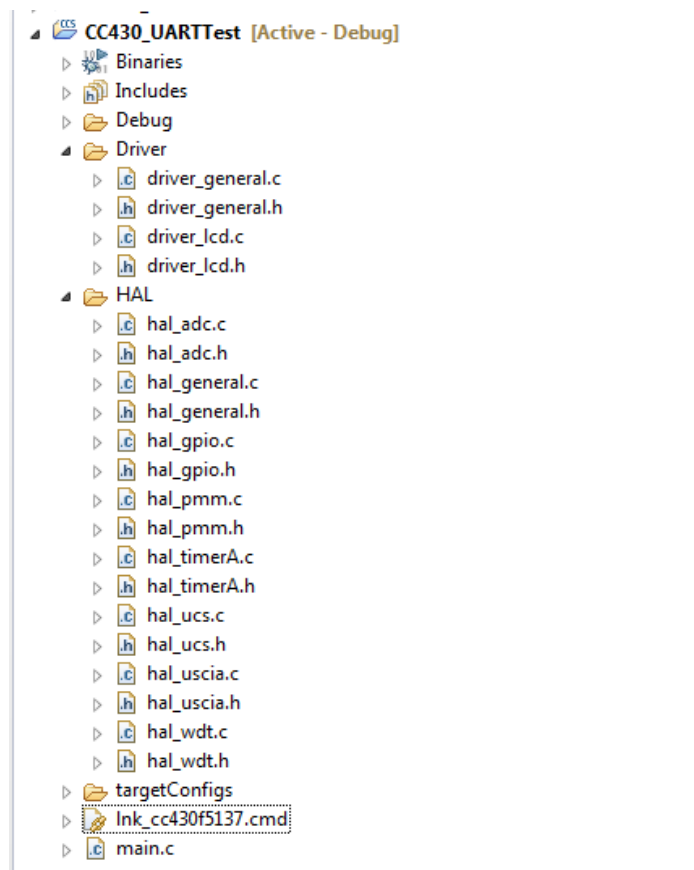


Abbildung 1: Screenshot Project Explorer CCS - Struktur

## 1.4. Code Composer Studio

Mithilfe des Code Composer Studios wird das Programm für den MSP430 kompiliert, gelinkt und per Debug Session und Spy-by-wire JTAG Schnittstelle auf die Hardware übertragen und dort dauerhaft im Flash Speicher abgelegt.

### Workspace

Ein Workspace kann mehrere Projekte beinhalten. Es empfiehlt sich den Workspace auf einem lokalen Datenträger z.B.: D zu speichern. Der Workspace kann dann im Ganzen auf das Netzlaufwerk kopiert werden. Vorsicht beim Kopieren einzelner Projekte.

### Projekt


Für jede Laborübung sollte ein neues Projekt angelegt werden. Da die Übungen aufeinander aufbauen, empfiehlt es sich das vorherige Projekt zu kopieren. Falls in einem Workspace mehrere Projekte angelegt sind, sollte man darauf achten, welches Projekt gerade editiert wird. Dies kann man dadurch erkennen, dass der Projektname des aktuellen Projektes fett angezeigt wird.


### Projekt erstellen

File -> New -> CCS Project

Target Name

Project Name

Build => 

Build, Download and Debug => 

### Ressource Explorer

View => Ressource Explorer => MSPWare: Sind Beispiele für jeden MSP430 zu finden.

### Edit / Debug Mode

Am rechten oberen Fensterrand befinden sich 2 Buttons für Edit und Debug Mode. Hier kann nur die Ansicht umgeschaltet werden.

### Optimizer

Um zu Beginn die Beeinflussung des Optimizers auszuschließen, muss dieser in jedem neu angelegten Projekt deaktiviert werden.

Rechtsklick auf das Projekt => Properties => Build => MSP430 Compiler => Optimization

Optimization level: off

Speed vs. Size trade-offs: 0

### Optional

Projektkontextmenü

Clean Project: Löscht alle Dateien die während eines Build-Vorgangs erstellt wurden.

## Softwarestruktur

Um das besprochene Layermodell umzusetzen, siehe Abbildung 1: Screenshot Project Explorer CCS - Struktur, müssen die entsprechenden Ordner angelegt werden.

Rechtsklick auf das Projekt => New => Folder

Ordner: HAL, DL, AL oder Hardware, Driver, Application o.ä.

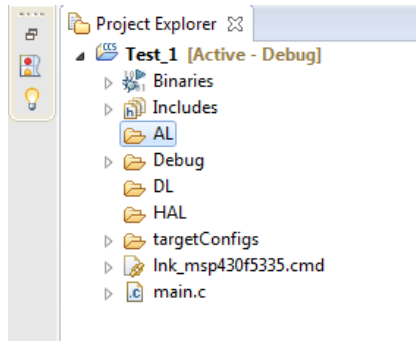


Abbildung 2: Screenshot Project Explorer – Software Basis Struktur

In jedem Ordner liegen nun die entsprechenden Dateien für den jeweiligen Layer. Als eindeutige Namenskonvention der C und H Dateien kann folgendes verwendet werden: hal\_modul.c, hal\_modul.h. z.B. hal\_usciB1.c, hal\_usciB1.h.

Jede C-Datei eines Moduls hat eine entsprechende Header Datei mit den jeweiligen Definitionen.

Möchten sie bereits fertige C-Dateien zum Projekt hinzufügen, kann man dies direkt in der Ordner Struktur vornehmen. Hier muss lediglich die Datei in das jeweilige Verzeichnis kopiert werden.

Würde man im main.c die hal\_usciB1.h inkludieren wollen, müsste man als Pfad „HAL/hal\_usciB1.h“ angeben. Um dies zu erleichtern kann man unter den Projekt Properties die include-Pfade angeben.

Damit durchsucht der Präprozessor nicht nur die Standardpfade, sondern auch die Layer-Pfade.

Rechtsklick Projekt => Properties => Build => MSP430 Compiler => Include Options.

Im unteren Textfeld auf das „+“ Symbol klicken und den jeweiligen Ordner im Workspace auswählen.

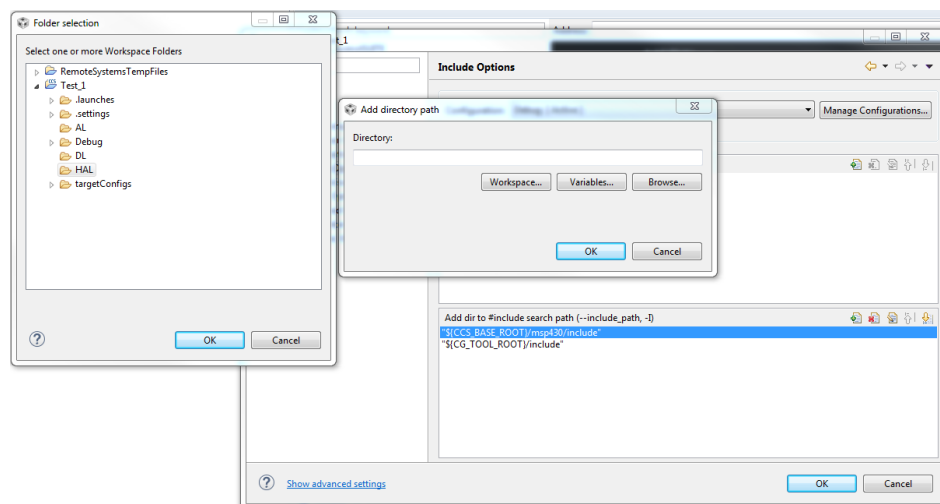


Abbildung 3: Screenshot Code Composer Studio – Include Optionen



### Debugger – In-Circuit Emulator

Klickt man auf das Download and Debug Symbol, buildet das Code Composer Studio das aktuelle Projekte und lädt es auf den Mikrokontroller in den Flash Speicher. Anschließend kann das Programm direkt auf der Hardware gestartet, gestoppt und an einer gewünschten Stelle mittels Breakpoint angehalten werden. Über den Register Viewer können Inhalte einzelner Register angezeigt bzw. direkt auf der Hardware geändert werden. Somit muss nicht zwingend das Programm umgeschrieben werden, wenn man eine andere Hardwarekonfiguration testen möchte.

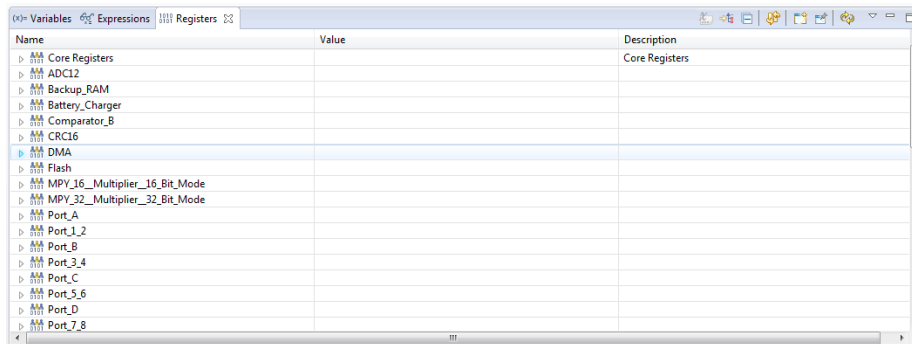


Abbildung 4: Screenshot Code Composer Studio – Register View

Weiters kann man Variablen im Variables bzw. Expression Fenster anzeigen. Damit hat man eine vollständige Online-Debug Funktionalität. Vorsicht: Einzelne Hardwarekomponenten wie z.B. Timer laufen auch bei angehaltenem Programm weiter. Möchte man dies verhindern, muss diese Option unter Tools => Debugger Options => MSP430 Debugger Options => Clock Control verändert werden.

## 1.5. Anlegen eines Basis Projektes

### Durchzuführende Arbeit

1. Legen sie ein neues Projekt für den verwendeten Mikrokontroller an: z.B. Laboruebung\_1
2. Erstellen sie die Layer Ordner und fügen sie die include-Pfade hinzu.
3. Schalten sie den Optimizer aus.
4. Fügen sie eine Endlosschleife der Main Funktion hinzu und tauschen sie das int gegen void aus.
5. Builden sie das Programm.
6. Kopieren sie die Dateien hal\_pmm.c und hal\_pmm.h in die Projektordner Struktur im Windows Explorer. Die Dateien scheinen dann automatisch im Project Explorer auf.
7. Erstellen sie eine neue C und H Datei: hal\_general.c, hal\_general.h  
Vergessen sie nicht in der Headerdatei die #ifndef Abfrage anzugeben.  
Inkludieren sie die Header Datei in der C Datei.
8. Erstellen sie in der hal\_general.c eine Funktion HAL\_Init ohne Übergabeparameter. Schreiben sie den Funktionsprototypen in die hal\_general.h Datei und inkludieren sie diese in der C Datei.
9. Inkludieren sie die hal\_pmm.h in der hal\_general.c Datei.
10. Rufen sie in der Funktion HAL\_Init die Funktion HAL\_PMM\_Init() auf.
11. Inkludieren sie die hal\_general.h in der main.c und rufen sie in der Main Funktion die Funktion HAL\_Init() auf.
12. Erstellen sie für die Konfiguration des Watchdog-Timers ein eigene Modul-C-Datei und erstellen sie eine Funktion HAL\_Wdt\_Init. Diese wird ebenfalls in der HAL\_Init Funktion aufgerufen, bevor das PMM Modul initialisiert wird.

## 1.6. Schnittstellen zwischen Softwaremodulen

Als Schnittstelle kann eine Funktion mit Übergabeparameter oder eine Datenstruktur dienen. Eine Schnittstelle muss **immer** eindeutig sein und darf nach der Definition nicht mehr verändert werden. Die Werte die der Schnittstelle übergeben werden, sollten für die Verbesserung der Lesbarkeit eindeutige Wertebereiche sein: z.B. Lenkung 0 – 100%, Abstand: 0 – 1000mm usw.

## 1.7. I/O Port Konfiguration

Jeder Mikrokontrollerpin muss je nach Aufgabe so konfiguriert werden, dass dieser entweder ein digitaler/analoger Eingang/Ausgang ist, oder direkt an ein Peripheriemodul angeschaltet wird, z.B. SPI MOSI. Um die Lesbarkeit des späteren Programms zu verbessern, benutzen sie sogenannte Makros, welche in der Headerdatei definiert werden. Hierzu können sie jedem Pin einen eigenen lesbareren Namen geben z.B. LCD\_BACKLIGHT. Weiters können sie einfache digitale Operationen als Makro angeben.

z.B. `#define LCD_BACKLIGHT_ON (P8OUT |= LCD_BL)`

Aufruf: `LCD_BACKLIGHT_ON;`

### Durchzuführende Arbeit & Dokumentation

1. Erstellen sie eine Modul-C-Datei „hal\_gpio.c“ und die entsprechende Headerdatei dazu.
2. Programmieren sie eine HAL\_GPIO\_Init() Funktion, welche alle I/O Pins als digitalen Ein- bzw. Ausgang definiert. Beachten sie zu diesem Zeitpunkt keine Analog- oder Peripheriefunktionen des Pins. Benutzen sie dazu Makros um die Lesbarkeit des Programmes zu gewährleisten.
3. Überlegen sie sich wie unbenutzte/nicht genutzte Port-Pins konfiguriert werden.
4. Rufen sie die HAL\_GPIO\_Init Funktion in der HAL\_Init Funktion auf und testen sie ihre Funktionalität mit dem Debugger.

## 2. Digitale Ein-/Ausgabe

### 2.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 1.3: Interrupts
- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 12: Digital I/O Module

#### Wissensüberprüfung

- Internet-Recherche: Was ist ein Interrupt in der Mikrokontroller Technik?
- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 12: Digital I/O Module
    - 12.2 Digital I/O Operation: 12.2.7 Port Interrupts

### 2.2. Rechendauer Performance Test

Nach Abschluss der I/O Port Konfiguration, überprüfen sie die Rechendauer einer Integer- bzw. einer Fließkommazahlenberechnung.

#### Durchzuführende Arbeit & Dokumentation

1. Führen sie eine Fließkommazahlen- und anschließend ein Integer Berechnung durch.
2. Schalten sie vor jeder Berechnung die Display Hintergrundbeleuchtung ein und nach der Berechnung wieder aus.
3. Zeigen sie am Oszilloskop die jeweilige Rechendauer an und dokumentieren sie diese.

### 2.3. Allgemeine I/O Interrupt-Steuerung

Das Programm soll das Drücken der Start bzw. Stoptaste per Interrupt und nicht mit *Polling* erkennen.

#### Durchzuführende Arbeit & Dokumentation

1. Erweitern sie ihre I/O Port Konfiguration so, dass sie die Interrupts für Start und Stoptaste einschalten.
2. Stellen sie die Flankentriggerung des jeweiligen Pins ein.
3. Was ist Kontaktprellen und wie kann mit dies schaltungstechnisch oder per Software vermeiden?
4. Inwieweit beeinflusst ein Interrupt den zeitlichen Ablauf ihres Programms?
5. Was ist *Interrupt Nesting*?

## 2.4. Interrupts

Wird ein Interrupt freigeschalten, springt das Programm bei Aktivierung des Interrupts auf die Adresse der Interrupt Service Routine – ISR die in der Interrupt Vector Tabelle gespeichert ist. Diese Funktion muss an dieser Speicherstelle vorhanden sein, da sonst der Program Counter – PC an einer ungültigen Speicherstelle die ISR sucht => Mikrokontroller stürzt ab.

### Durchzuführende Arbeit & Dokumentation

1. Schreiben sie eine ISR für den Port Vektor. Die korrekte ISR-Syntax entnehmen sie einem Example aus dem **Ressource Explorer**
2. Aktivieren sie das GIE Bit am Ende der HAL\_GPIO\_Init Funktion => siehe Example
3. Typdefinieren sie eine Datenstruktur in der hal\_gpio.h nach folgender Vorgabe:

```
typedef struct {  
    unsigned char active;           // TRUE 1 / FALSE 0  
    unsigned char button;          // Button number  
}ButtonCom;
```

4. Deklarieren sie eine globale Variable vom Typ ButtonCom in der hal\_general.c.
5. Deklarieren sie diese Variable vom Typ ButtonCom in der main.c und hal\_gpio.c als externe globale Variable.
6. Fragen sie in der ISR ab welche Taste gedrückt worden ist, unter der Berücksichtigung, dass nur 2 Tasten vorhanden sind => Bitmask.
7. Setzen sie das active flag und die button Nummer in der ISR.
8. Fragen sie das active Flag und die button Nummer in der main-Endlosschleife ab. Wenn die Starttaste gedrückt wird soll die Hintergrundbeleuchtung eingeschaltet werden. Wenn die Stoptaste gedrückt worden ist soll die Hintergrundbeleuchtung ausgeschaltet werden.
9. Welche Interrupt Priorität besitzt der verwendete Port?
10. Welche Funktion besitzt das GIE Bit?

## 3. Clock System, TimerB0 Konfiguration

### 3.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 5: Unified Clock System
- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 18: Timer B
- Crazy Car Controller FHJ Schaltplan (Quarz, Clock)

#### Wissensüberprüfung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 5.1: Unified Clock System (UCS) Introduction
    - Clock Sources
    - UCS Clock Signals
- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 18.1: Timer B Introduction
    - Timer Bit Breite
    - Zusammenhang: Eingangsfrequenz => Timer Teilungswert => Ausgangsfrequenz
- Taschenrechner für Wissensüberprüfung

### 3.2. Power Management Module

Das PMM wird in dieser Laborübung nicht behandelt, es wird aber trotzdem eine Konfiguration benötigt. Um die höchste Taktrate von 20MHz nutzen zu können, binden sie die `hal_pmm.c` und die Header Datei ein, siehe Anlegen eines Basis Projektes.

### 3.3. Unified Clock System

Um die Peripherie im Mikrokontroller nutzen zu können, muss eine entsprechende Taktquelle ausgewählt und konfiguriert werden. Damit der Mikrokontroller/CPU hochfahren kann, ist per Default eine Taktquelle eingestellt. Die Anforderungen an die Taktquelle ergeben sich aus der Frequenz, Genauigkeit sowie Energieverbrauch des Mikrokontrollers. Zuerst muss die Taktquelle ausgewählt werden, z.B. Quarz. Ist ein Quarz an den jeweiligen Pins angeschlossen, müssen diese entsprechend konfiguriert werden. Jede interne Taktquelle MCLK, SMCLK, etc. muss konfiguriert und entsprechend eingestellt werden.

#### Durchzuführende Arbeit & Dokumentation

1. Konfigurieren sie die XT2 Pins in der `hal_gpio.c` so um, dass diese den Quarz treiben können und ans UCS Modul angeschlossen sind.
2. Erstellen sie ein neues HAL Modul, `hal_ucs.c` und `hal_ucs.h`, und schreiben sie eine Funktion `HAL_UCS_Init()`. Diese soll in der `HAL_Init` Funktion aufgerufen werden.
3. Konfigurieren sie das UCS Modul so, dass der Quarz die Taktquelle für alle Ausgangstakte ist.
4. Entnehmen sie die Konfiguration des UCS Moduls einem Example im Ressource Explorer.
5. Berücksichtigen sie bei der Konfiguration die *Oscillator Fault Flags*.
6. Konfiguration:  
MCLK = 20MHz  
SMCLK = 2.5MHz  
ACLK = n.d.
7. Kontrollieren und Dokumentieren sie die Frequenz von SMCLK mit dem Oszilloskop am entsprechenden Pin. Dazu schalten sie die Funktion des Pins in der GPIO-Konfiguration frei.
8. Definieren sie die konfigurierten Frequenz im Header File für die spätere Verwendung, z.B.:  

```
#define XTAL_FREQU 20000000  
#define MCLK_FREQU 20000000  
etc.
```

### 3.4. Timer Interrupts – Timer B0

Applikationen die auf Mikrokontrollern ablaufen erfordern meist ein genaues Timing. Dieses Timing bzw. zyklische Aufrufen von Funktionen/Routinen/Triggern kann durch einen Timer erreicht werden. Diese Timer sind ein Teil der Peripherie des Mikrokontrollers und laufen in Hardware => verbrauchen somit keine CPU Rechenzeit. Der TimerB0 soll so konfiguriert werden, dass die Hintergrundbeleuchtung des Displays im 2Hz Takt blinkt. Verwenden sie dazu das Capture/Compare Register 0 – TBCCR0.

#### Durchzuführende Arbeit & Dokumentation

1. Erstellen sie ein neues HAL Modul, `hal_timerB0.c` und `hal_timerB0.h`, und schreiben sie eine Funktion `HAL_TimerB0_Init()`. Diese soll in der `HAL_Init` Funktion aufgerufen werden.
2. Konfigurieren sie den Timer entsprechend der Vorgaben. Schalten sie die Hintergrundbeleuchtung des Displays mittels Makro in der ISR des Timers ein bzw. aus; Tipp: Toggle I/O Pin.
3. Benutzen sie für die Berechnung des Timer Wertes die im UCS Modul definierten Werte. Definieren sie auch die Ausgangsfrequenz des Timers mittels `define`.
4. Fragen sie in der ISR das Richtige Interrupt Flag des Timers ab.
5. Kontrollieren und Dokumentieren sie die Frequenz der Hintergrundbeleuchtung mit dem Oszilloskop.

## 4. PWM Erzeugung, Aktorik

### 4.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 17: Timer A
- CrazyCar\_HPI-ESC\_Modified\_Manual
- Crazy Car Controller FHJ Schaltplan

#### Wissensüberprüfung

- Internet-Recherche:
  - Was ist ein PWM Signal?
  - Was ist der Duty Cycle?
  - Standard Servo-Signal im Modellbau
- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 17.1: Timer A Introduction
    - Zusammenhang: Eingangsfrequenz => Timer Teilungswert => PWM Auflösung
  - Chapter 17.2.4.2: Compare Mode
    - Output Modes
    - Output Example Timer in Up Mode

### 4.2. Timer A1 – PWM Erzeugung

Es soll ein pulswidenmoduliertes Signal mittels Timer A1 erzeugt werden, mit welchem das Lenkservo bzw. der elektronische Fahrtenregler angesteuert wird. Die Frequenz des Signals soll 60Hz betragen. Die Pulsbreite, im Bereich von ca. 1ms bis 2ms, gibt vor auf z.B. welche Position sich das Lenkservo bewegt bzw. wie schnell sich der Motor drehen soll. **VORSICHT:** Das Servo darf nicht weiter drehen, als es die Lenkung mechanisch zulässt.

#### Faustregel

1ms    ganz links/rechts  
1,5ms    mitte  
2ms    ganz rechts/links

#### Durchzuführende Arbeit & Dokumentation

1. Erstellen sie ein neues HAL Modul, hal\_timerA1.c und hal\_timerA1.h, und schreiben sie eine Funktion HAL\_TimerA1\_Init(). Diese soll in der HAL\_Init Funktion aufgerufen werden.
2. Konfigurieren sie den Timer entsprechend der Vorgaben, für 2 PWM Kanäle: Frequenz 60Hz => Definitionen. Dabei muss eine geeignet große Timer Eingangsfrequenz gewählt werden, um eine genaue Auflösung der einzelnen Bereiche, z.B. Bereich Links 1ms – 1.5ms, zu erreichen.
3. Startbedingung: Der Timer soll nach der Initialisierungsfunktion keinen Puls erzeugen: Duty Cycle 0%.
4. Die jeweiligen GPIO Pins müssen für die Verwendung als Timer Output konfiguriert werden.



5. Servo
6. Schließen sie das Servo am Crazy Car Controller an. Die Versorgung für das Servo wird über den Fahrtenregler bereitgestellt. Stecken sie den Akku an den Fahrtenregler und schalten sie diesen ein. VORSICHT: Das Fahrzeug muss aufgebockt werden, sollten sich im Fehlerfall die Räder bewegen.
7. Ermitteln sie die Position für: Mittelstellung, max. Links und max. Rechts in dem sie im Debugger die Werte der einzelnen PWM Kanal Register ändern.
8. „Speichern“ sie diese Werte als Definition im Header File ab.
9. Fahrtenregler ESC
10. Der Fahrtenregler muss vor jeder Verwendung konfiguriert und parametrisiert werden. Beachten sie die Konfigurationsroutine für den PWM Modus im Dokument *CrazyCar\_HPI-ESC\_Modified\_Manual*. Diese Konfiguration soll im Driver Layer stattfinden.

#### 4.3. Driver Layer - Aktorik

Um eine Abstraktion zum Hardware Layer - Aktorik zu erreichen, werden verschiedene Schnittstellen definiert. In diesem Fall ist die Schnittstelle eine/mehrere Funktionen von der aus die Applikation auf die Hardware/Aktorik zugreifen kann. Diese Schnittstelle darf nicht mehr verändert werden.

##### Durchzuführende Arbeit & Dokumentation

1. Erstellen sie einen Ordner Driver in der Projekthierarchie.
2. Erstellen sie ein neues Driver Modul, driver\_general.c und driver\_general.h
3. Programmieren sie eine Funktion Driver\_Init(void) in der driver\_general.c. Hier werden alle Driver Module konfiguriert, vgl. Anlegen eines Basis Projektes.
4. Erstellen sie ein neues Driver Modul, driver\_aktorik.c und driver\_aktorik.h
5. Programmieren sie eine Funktion Driver\_SetSteering. Als Übergabeparameter soll ein Wertebereich gewählt werden, der einfacher verwendbar und gleichbleibend ist.  
z.B.  $-100 \div 0 \div +100$ ,  $-100 \triangleq \text{max. Links}$ ,  $+100 \triangleq \text{max. Rechts}$ ,  $0 \triangleq \text{Mitte}$
6. Programmieren sie eine Funktion Driver\_SetThrottle. Diese soll ähnlich wie die Funktion SetSteering funktionieren. Beachten sie dabei das *CrazyCar\_HPI-ESC\_Modified\_Manual*  
Beide Funktionen, SetSteering und SetThrottle müssen den übergebenen Wert auf den jeweiligen Registerwert umrechnen. Diese Werte bilden die standardisierte Schnittstelle.
7. Programmieren sie eine Funktionen Driver\_ESCInit oder Driver\_ThrottleInit, welche den ESC konfiguriert bzw. parametrisiert, siehe *CrazyCar\_HPI-ESC\_Modified\_Manual*. Die jeweiligen Wartezeiten können sie mit dem PWM Timer oder mit der Funktion `__delay_cycles()` lösen. Rufen sie die Initialisierungsfunktion des Fahrtenreglers in der Driver\_Init Funktion auf.

#### 4.4. Drehzahlmessung (optional)

Realisieren sie die Drehzahlmessung mit Hilfe der Capture-Funktion des Timers.

## 5. SPI

### 5.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 37: USCI – SPI Mode
- SPI: Signale, Aufbau und Funktionsweise
- Crazy Car Controller FHJ Schaltplan

#### Wissensüberprüfung

- Internet-Recherche:
  - SPI: Signale, Aufbau und Funktionsweise
- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 37.1: USCI Overview
  - Chapter 37.2: USCI Introduction – SPI Mode / Block Diagram
  - Chapter 37.3: USCI Operation Mode

### 5.2. USCI B1 - Konfiguration

Eine SPI Schnittstelle wird auf der Crazy Car Hardware dazu benötigt um das LC-Display entsprechend zu benutzen und etwas anzuzeigen. Das Display ist auf der USCI\_B1 angeschlossen und die Pins müssen entsprechend konfiguriert werden – I/O Module. Ebenso muss die USCI B1 Schnittstelle im SPI Mode konfiguriert und auf das Übertragungsprotokoll des Displays eingestellt werden. Es soll, mittels HAL Funktion, ein oder mehrere Bytes zum Display übertragen werden können. Die Übertragung soll mittels ISR Funktion realisiert werden, um die CPU so wenig wie möglich zu belasten. Die Daten, die übertragen werden, müssen in die TxData Datenstruktur gespeichert werden, anschließend wird mittels einmaligem Funktionsaufruf die Übertragung getriggert. Das *Chip Select* Signal kann vom MSP430 SPI Module nicht automatisch erzeugt werden. Dieses muss, am besten per Makro z.B.: CS\_LOW, manuell gesetzt oder rückgesetzt werden.

#### Durchzuführende Arbeit & Dokumentation

1. Erstellen sie ein neues HAL Modul, hal\_usciB1.c und hal\_usciB1.h
2. Programmieren sie eine Funktion HAL\_USCIB1\_Init(void). Diese Funktion soll die USCI B1 Schnittstelle wie folgt konfigurieren:

*SPI Master Mode*

*100 kHz clock frequency => Als define in der Header Datei*

*8 bit, MSB first*

*Clock Phase: 0*

*Clock Polarity: 1*

Schalten sie den RX Interrupt der Schnittstelle frei.

3. Nehmen sie die entsprechenden Konfigurationsänderungen im GPIO Modul vor.
4. Typdefinieren sie folgende Datenstruktur in der hal\_usciB1 Header Datei:

```
typedef struct {
    union{
        unsigned char R;
        struct {
            unsigned char TxSuc:1;    // Bit=1 wenn Daten übertragen wurden
            unsigned char dummy:7;
        }B;
    }Status;

    struct {
        unsigned char len; // Länge der Daten in Bytes die übertragen werden
        unsigned char cnt; // Index auf momentan übertragene Daten
        unsigned char Data[256]; // Tx Daten Array
    }TxData;

    struct {
        unsigned char len; // Länge der empfangenen Daten
        unsigned char Data[256]; // Rx Daten Array
    }RxData;
}USCIB1_SPICom;
```

5. Deklarieren sie eine globale Variable vom Typ USCIB1\_SPICom in der hal\_general.c.
6. Deklarieren sie diese Variable vom Typ USCIB1\_SPICom in der main.c und hal\_usciB1.c als externe globale Variable.
7. Programmieren sie eine Funktion HAL\_USCIB1\_Transmit. Diese Funktion soll das erste Byte in den TXBUF schreiben. Alle weiteren Bytes werden von der ISR in den TXBUF geschrieben bis die Länge len erreicht wird. Bei Aufruf dieser Funktion, soll das TxSuc Bit auf 0 gesetzt werden.
8. Programmieren sie eine ISR Funktion, welche den RX Interrupt dazu benutzt um ein weiteres Byte zu senden. Es muss der RX Interrupt abgefragt werden, weil eine 1 Byte SPI Übertragung erst nach dem Empfang des Bytes abgeschlossen ist.  
Dabei ist die Abbruchbedingung der Vergleich zwischen der Länge der Daten und der Wert in der Cnt Variable in der TxData Struktur. Ist eine SPI Übertragung vollständig abgeschlossen, setzen sie das TxSuc Bit auf 1. Dieses Bit wird im Programmablauf dazu verwendet, um eventuell abzuwarten bis die SPI Übertragung vollständig ausgeführt worden ist, z.B. Initialisierung des Displays.
9. Jedes empfangene Byte soll in das RxData Datenarray geschrieben werden.
10. Arbeiten sie das *Chip Select* Signal entsprechend der Vorgaben in die Übertragung mit ein.
11. Kontrollieren und Dokumentieren sie alle SPI Signale mit dem Oszilloskop und stellen sie eine 2 Byte Übertragung dar.

### 5.3. USCIA0 - UART Konfiguration (optional)

Das onPCB Debugtool besitzt neben dem Debug Interface eine virtuelle COM Schnittstelle, mit der man mit dem Crazy Car Controller vom PC aus über USB kommunizieren kann. Dazu muss die USCIA0 Schnittstelle im UART Mode betrieben und wie folgt eingestellt werden:

*Übertragungsrate: 9600 Baud*

*1 Stop Bit*

*8 Datenbits*

Um nun Daten mit dem Controller auszutauschen, kann das Programm Hterm benutzt werden. Dies ist ein einfaches Terminal Programm mit dem eine Verbindung über ein COMPORT aufgebaut werden kann.

## 6. SPI

### 6.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 37: USCI – SPI Mode
- SPI: Signale, Aufbau und Funktionsweise
- Crazy Car Controller FHJ Schaltplan
- Fertigstellung der Übung aus Kapitel 5. SPI
- Beginn der Übung aus Kapitel 7. LC - Display

#### Wissensüberprüfung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 37.3.6: Serial Clock Control
  - Chapter 37.3.8: USCI Interrupts in SPI Mode

## 7. LC - Display

### 7.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments  
Chapter 37: USCI – SPI Mode
- ST7565 65 x 132 Dot Matrix LCD Controller Datasheet - ST7565.pdf
  - Page 7: Block Diagram
  - Page 12 – 15: MPU Interface
  - Page 16: Font Definition
  - Page 32 – 41: Commands
- Crazy Car Controller FHJ Schaltplan

#### Wissensüberprüfung

- ST7565 65 x 132 Dot Matrix LCD Controller Datasheet - ST7565.pdf
  - Page 13: The Serial Interface - MPU Interface
  - Page 15: Display Data RAM
  - Page 16: Font Definition

### 7.2. Funktionen des Displays

Das Display besitzt eine Auflösung von 128x64 Pixel. Jeder einzelne Pixel kann aus bzw. eingeschaltet werden. Eine simple Datenübertragung mittels SPI genügt hier nicht um auf dem Display etwas anzuzeigen. Es muss nicht nur zwischen dem Kommando und Datenmodus umgeschaltet werden, sondern auch eine Initialisierung des Displays muss stattfinden. Ebenso muss das Kommunikationsinterface des Displays mit Chip Select laut Datenblatt aktiviert werden.

Wichtige Kommandos für das Display per SPI (Datenblatt Seite 41)  
Diese sollten als Definition in der Headerdatei abgelegt werden.

LCD_RESET	0xD2	
LCD_BIAS	0xA3	
ADC_SEL_NORMAL	0xA0	
COMMON_REVERSE	0xC8	
RES_RATIO	0x24	
ELEC_VOL_MODE	0x81	
ELEC_VOL_VALUE	0x0F	Kontrast Einstellung
POWER_CONT	0x2F	
DISPLAY_ON	0xAF	

### 7.3. Display Driver

Das Display ist elektrisch mit der SPI Schnittstelle des Mikrokontrollers verbunden. Die Funktionen für die Übertragung der Daten über SPI aus der Übung 5 und 6 werden im Driver des Displays aufgerufen. Als Hilfestellung werden die Low-Level Driver Funktionen für Textausgabe die sie mindestens benötigen in den folgenden Punkten beschrieben.

### Durchzuführende Arbeit & Dokumentation

1. Erstellen sie ein neues Driver Modul: driver\_lcd.c, driver\_lcd.h und binden sie es entsprechend ein.

2. **Driver\_LCD\_WriteCommand (unsigned char \*data, unsigned char len)**

Schreiben sie eine Funktion, welcher sie einen Pointer auf ein Array von Daten in Bytes und die Anzahl an Bytes der Daten übergeben. Diese Funktion schaltet den Data/Command Pin auf Kommando um, kopiert die Daten ins SPI Datenarray und überträgt die Daten ans Display.

3. **Driver\_LCD\_Init()**

Schreiben sie eine Funktion, welche das Display initialisiert. Diese Funktion soll im Driver\_Init() aufgerufen werden.

Gehen sie dabei wie folgt vor.

LCD Backlight ON (optional)

Reset Line LOW

\_\_delay\_cycles(100000); Warte auf Display

Reset Line HIGH

Send Command

{ LCD\_RESET, LCD\_BIAS, ADC\_SEL\_NORMAL, COMMON\_REVERSE, RES\_RATIO,  
ELEC\_VOL\_MODE, ELEC\_VOL\_VALUE, POWER\_CONT, DISPLAY\_ON }

\_\_delay\_cycles(100000); Warte auf Display

LCD Clear: Reset all Pixels on Display

TIPP: Führen sie einen Display Test am Ende durch, indem sie alle Pixel des Displays einschalten. Damit können sie darstellen, ob alle Pixel ordnungsgemäß funktionieren.

4. **Driver\_LCD\_SetPosition(unsigned char page, unsigned char col)**

Programmieren sie eine Funktion, welche die Position des Cursors am Display setzt. Von dieser Cursorposition aus, wird der z.B. Text auf das Display geschrieben. Die Spalte kann dabei als Pixel oder Zeichenwert interpretiert werden. Die Zeilen und Spaltenwerte müssen als Kommando ans Display gesendet werden. Detailinformationen entnehmen sie dem Datenblatt des Displaycontrollers.

5. **Driver\_LCD\_Clear(void)**

Programmieren sie eine Funktion, welche jeglichen Inhalt vom Display löscht. Dies wird benötigt um nach der Initialisierung den kompletten Display RAM zu löschen.

## 6. **Font Table**

Um Buchstaben, Zahlen und Zeichen am Display darstellen zu können, muss eine entsprechende Zeichensatztabelle erstellt werden. Jedes Zeichen besteht aus mehreren einzelnen Pixeln. Die Tabelle wird als mehrdimensionales globales Array im Flashspeicher (const) abgelegt. Der jeweilige Index jedes Zeichens soll dabei dem ASCII Wert des Zeichens entsprechen. Wie man diese Zeichentabelle erstellt, entnehmen sie dem Datenblatt des Displaycontrollers.

## 7. ***Driver\_LCD\_WriteText (unsigned char \*text, unsigned char len, unsigned char page, unsigned char col)***

Programmieren sie eine Funktion, die den übergebenen Text mit der Zeichenlänge len am Display darstellt. Setzen sie vorher laut page und col Variable den Cursor an die entsprechende Position des Displays. Es müssen für jedes Zeichen, entsprechend der Font Tabelle, die Pixelwerte an das Display im Datenmodus übertragen werden.

## 8. ***Driver\_LCD\_WriteUInt (unsigned int num, unsigned char page, unsigned char col)***

Diese Funktion soll ähnlich wie die WriteText Funktion arbeiten. Hier muss jedoch die positive Integer Zahl zuerst in Text umgewandelt werden. Dannach müssen dieselben Schritte wie in der WriteText Funktion durchgeführt werden. Die WriteText Funktion kann auch in dieser Funktion benutzt werden.

## 7.4. Fast Display Driver (optional)

Um die CPU zu entlasten kann die Funktionalität, wie sie in Grafikkarten benutzt wird, umgesetzt werden. Dazu wird ein zweidimensionales Array deklariert - das Display RAM - welches eine exakte Spiegelung des Display Inhaltes darstellt. Funktionen die am Display etwas anzeigen, WriteText, WriteUInt, schreiben nicht mehr direkt ans Display sondern in dieses Display RAM. Dieses wird mittels DMA über SPI ans Display übertragen. Somit wird die CPU mit der Übertragung der Daten nur mehr sehr gering belastet.

**Nachteil:** Das Grafik RAM verbraucht sehr viel Platz im Arbeitsspeicher des Mikrokontrollers.



## 8. ADC

### 8.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 28: ADC12\_A
- Crazy Car Controller FHJ Schaltplan

#### Wissensüberprüfung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 28: Figure 28-1 – ADC12 Block Diagram
  - Chapter 28.2.7.1 – 28.2.7.4: Funktionsweise der Modi (Zeichnen von Flow Charts nicht notwendig)
  - Allgemein: Zusammenhang von Bit-Auflösung, Referenzspannung und max. Eingangsspannung
- Taschenrechner für Wissensüberprüfung

### 8.2. ADC Konfiguration

Der Analog-Digital Konverter wird benötigt um die analoge Spannung der Abstandssensoren und der Versorgungsspannung einzulesen und in einen verarbeitbaren Wert umzuwandeln. Der MSP430 besitzt einen 12 Bit SAR ADC. Dieser kann auf mehreren Input Pins mittels Analogmultiplexer Spannungen sequentiell umwandeln. Die umgewandelten Werte können dann im dementsprechenden MEM Register bzw. per DMA gelesen und gespeichert werden.

#### **Sample time (SAMPCON High)**

Die Sample Time legt fest wie lange das Eingangssignal am Eingangspin anliegt. Das „Einlesen“ passiert über einen analogen Multiplexer, welcher das Eingangssignal auf einen RC Kreis schaltet. Dieser wird dann mit der Eingangsspannung aufgeladen, vgl. MSP430x5xx and MSP430x6xx Family User Guide Rev. O: Chapter 20.2.5.3. – Figure 28-6. Da die Eingangsbeschaltung ein RC Tiefpassfilter darstellt, muss die sample time entsprechend der Zeitkonstante lang genug gewählt werden, damit sich der Kondensator aufladen kann. Die Sample Frequenz legt fest, wie oft das Signal pro Sekunde eingelesen wird.

#### **Hold time (SAMPCON Low)**

In der „hold time“ wird der Multiplexschalter zu den externen Pins geöffnet und die „gespeicherte“ Spannung im Kondensator an den Eingang des ADCs gelegt. Anschließend erfolgt die Wandlung in 13 Taktzyklen. Die Dauer hängt dabei von der gewählten ADC Arbeitsfrequenz ab.

Das SAMPCON Signal sollte immer mit einer geeignet genauen Triggerquelle angesteuert werden. Das Signal selbst kann auf mehrere Arten und mit mehreren Quellen erzeugt werden: z.B. Timer, vgl. MSP430x5xx and MSP430x6xx Family User Guide Rev. O: Chapter 20.2.5.3. – Figure 28-2 ADC12SHSx. Hier kann man festlegen ob die Triggerquelle eine komplette Sequenz triggert und der Sample Timer im ADC die weiteren Pulse erzeugt oder die Quelle direkt das Sample Signal SAMPCON darstellt.

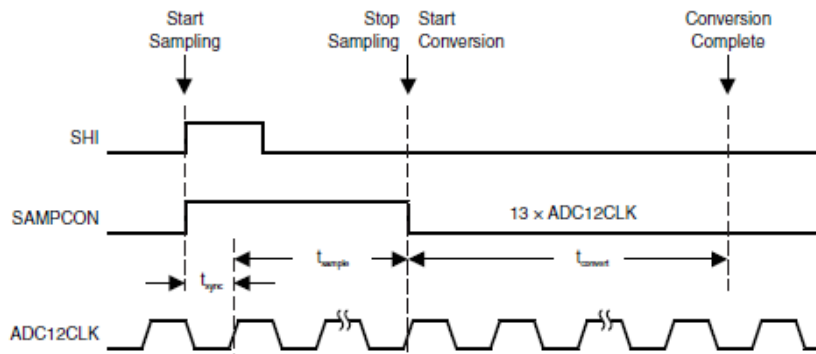


Abbildung 5: ADC12 Sample Timing

#### Durchzuführende Arbeit & Dokumentation

1. Erstellen sie ein neues HAL Modul, hal\_adc12.c und hal\_adc12.h
2. Konfigurieren sie die entsprechenden Pins im GPIO Modul als analogen Eingang.
3. Konfigurieren sie den Timer B0 so um, dass dieser den ADC12 triggert (Sequenz oder SAMPCON). Die Samplefrequenz soll 120 Hz betragen.
4. Programmieren sie eine Funktion HAL\_ADC12\_Init(void). Diese Funktion konfiguriert den ADC12. Folgende Dinge müssen dabei berücksichtigt werden:
  - Referenzspannung  $\leftrightarrow$  Eingangsspannung
  - Sample Frequency  $\leftrightarrow$  Frequenz des Eingangssignals
  - Sample-and-hold time
  - Arbeitsfrequenz des ADCs
  - Zielspeicher der gesampelten Werte
5. Typdefinieren sie folgende Datenstruktur in der hal\_adc12 Header Datei:

```
typedef struct {
    union{
        unsigned char R;
        struct {
            unsigned char ADCrdy:1;    // Bit=1 wenn Daten übertragen wurden
            unsigned char dummy:7;
        }B;
    }Status;

    unsigned int ADCBuffer[4];        // Speicherplatz der ADC Werte
}ADC12Com;

Oder:
    unsigned char SensorLeft;
    unsigned char SensorRight;
    unsigned char SensorFront;
    unsigned char VBat;
```

6. Deklarieren sie eine globale Variable vom Typ ADC12Com in der hal\_general.c.
7. Deklarieren sie diese Variable vom Typ ADC12Com in der main.c und hal\_usciB1.c als externe globale Variable.
8. Programmieren sie eine ISR für den ADC12, die nach erfolgreicher Sample Sequenz die Werte aus den MEM Registern in die globale Datenstruktur schreibt.
9. Setzen sie in der ISR, nach erfolgreichem Schreiben der Daten, das ADCrdy flag.
10. Fragen sie dieses Flag in der Endlos Main Schleife ab und schreiben sie die Daten auf das Display. Vergessen sie nicht das Flag rückzusetzen.

## 9. ADC - DMA

### 9.1. Inhalt

#### Laborübung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 28: ADC12\_A
  - Chapter 11: Direct Memory Access - DMA
- Crazy Car Controller FHJ Schaltplan
- Fertigstellung der Übung aus Kapitel 8. ADC

#### Wissensüberprüfung

- MSP430x5xx and MSP430x6xx Family User Guide Rev. O - Texas Instruments
  - Chapter 11.1: Introduction
  - Chapter 11.2.1: Addressing Modes
  - Chapter 11.2.2: Transfer Modes

### 9.2. ADC - DMA Konfiguration

Um die CPU mit dem Datentransfer des ADCs nicht zu belasten, wird ein Kanal des DMA Kontrollers eingesetzt. Dieser muss so konfiguriert werden, dass er nach Fertigstellung der Sample Sequenz die Werte automatisch aus den MEM Registern in die globale Datenstruktur schreibt.

1. Erstellen sie ein neues HAL Modul, `hal_dma.c` und `hal_dma.h`
2. Programmieren sie eine Funktion `HAL_DMA_Init(void)`. Diese Funktion konfiguriert den DMA0 Kanal soweit vor, dass dieser für den Einsatz beim ADC vorbereitet ist. Alternativ kann dies auch in der Initialisierungsfunktion des ADCs geschehen, allerdings ist dann aus der Programmstruktur nicht ersichtlich wofür der DMA0 eingesetzt wird.
3. Schreiben sie die Initialisierungsfunktion des ADCs so um, dass die Werte nach Beendigung der Sample Sequenz mit dem DMA in den globalen Speicher übertragen werden. Dabei soll nach erfolgreichem Übertragen wieder das `ADCrdy` Flag gesetzt werden, um der main Funktion mitzuteilen, dass neue Sensorwerte eingelesen wurden.

## 10. Sharp Abstandssensoren

### 10.1. Inhalt

#### Laborübung

- Sharp GP2Y0A Series/GP2Y0D Series Application Note / gp2y0a\_gp2y0d\_series\_appl\_e.pdf
- Sharp GP2Y0A02YK0F Datasheet / GP2Y0A02YK0F.pdf
- Sharp GP2Y0A21YK0F Datasheet / GP2Y0A21YK0F.pdf
- Crazy Car Controller FHJ Schaltplan

#### Wissensüberprüfung

- Sharp GP2Y0A Series/GP2Y0D Series Application Note - gp2y0a\_gp2y0d\_series\_appl\_e.pdf
  - Page 2 / Attachment-2: Measuring principle
  - Page 9 / Attachment-9: distance characteristics
- Sharp GP2Y0A02YK0F Datasheet / GP2Y0A02YK0F.pdf
  - Page 4: Timing Chart => Zusammenhang mit Sample Time
  - Page 5: Distance characteristics
- Taschenrechner für Wissensüberprüfung

### 10.2. Sensor Kennlinie

Bevor sie mit einem Sensor arbeiten können, müssen sie zuerst seine Charakteristik kennen und testen. Dies beinhaltet Versorgungsspannung, Spannungsbereich des Ausgangssignals und die Frequenz. Anhand dieser Daten können sie den Sensor optimal nutzen und gegebenenfalls mathematische Korrekturen der Signale im Mikrokontroller vornehmen bevor diese weiterverarbeitet werden.

1. Ermitteln sie die Kennlinie der Sensoren in dem sie den Abstand zu einem geeignet großen Objekt verändern und die Ausgangsspannung mit einem Multimeter messen. Stellen sie die Kennlinie grafisch dar, vgl. Sharp GP2Y0A02YK0F Datasheet / GP2Y0A02YK0F.pdf Page 5: Distance characteristics
2. Prüfen und dokumentieren sie mit dem Oszilloskop ob die Ausgangsspannungen der Sensoren wirklich linear sind. Was schließen sie daraus?
3. Linearisieren sie die Ausgangskennlinie des Sensors um einen linearen Zusammenhang zwischen Entfernung und Ausgangsspannung zu erreichen. Dies wird benötigt um einen linear arbeitenden Fahralgorithmus einsetzen zu können. Die Linearisierung kann auf zwei Arten durchgeführt werden:
  - (1) *Mathematische Annäherung*  
Bilden sie mittels mathematischer Funktion die Kennlinie im Mikrokontroller nach und schließen sie so auf den Abstand zurück.
  - (2) *Lookup Table*  
Legen sie für geeignet viele Messpunkte die Abstandswerte in einer Lookup Tabelle im Speicher des Kontrollers ab und schließen sie damit auf den Abstand zurück.
4. Erweitern sie ihr Programm so, dass die linearisierten Sensorwerte global abgespeichert werden. Dies kann durch ein weiteres Driver Layer Modul geschehen, welches die ADC Werte entgegennimmt und diese linearisiert abspeichert. Wertebereich z.B. 0 – 1500, 0 – 800.

## 11. Fahralgorithmen

### 11.1. Allgemein

Ziel der nun folgenden Laborübungen ist es, das Fahrzeug auf einer Rennstrecke autonom fahren zu lassen. Hier ist es notwendig eine Logik oder Regelung entscheiden zu lassen, wie sich das Fahrzeug durch den Kurs bewegen soll. Das Fahrzeug soll vorrangig sicher die Runden absolvieren können, dabei ist die Geschwindigkeit nicht vorrangig. Informationen, Ablauf und Reglement über den Crazy Car Bewerb finden sie auf der Crazy Car Homepage <https://fh-joanneum.at/projekt/crazycar> Der Bewerb findet am 2.2.2016 im Audimax Graz statt. Bitte melden sie sich früh genug mit ihrem Team auf der Homepage an.

Wir freuen uns auf ein spannendes Rennen!

### 11.2. Sensoren

#### **Abstandssensoren**

Die Abstandssensoren sind anfällig auf direktes Licht. Deshalb werden die Lichtverhältnisse im Audimax angepasst. Ebenso ist es wichtig den Winkel der Sensoren zur Fahrbahn bzw. zur Bande dementsprechend anzupassen, damit die Sensoren nicht den Boden bzw. über die Banden messen. Je steiler der Winkel zum Fahrzeug der vorderen Sensoren ist, desto besser „sehen“ die Sensoren in die Kurve aber schlechter Hindernisse vor dem Fahrzeug.

Da es bei der Messung immer wieder zu „Ausreißern“ kommen kann, empfiehlt es sich die Sensorwerte mittels „Sliding Averager“ zu filtern. Hier nimmt man einen oder mehrere Sensorwerte aus der Vergangenheit und mittelt diese mit dem gerade gemessenen Wert. Dadurch entsteht eine Filterwirkung die einzelne falsche Messwerte herausfiltert. Dies verschlechtert jedoch die Reaktionszeit des Sensors bei momentanen Ereignissen.

#### **Drehzahlsensor**

Das Verhalten des Fahrzeugs hängt sehr stark von der Geschwindigkeit ab. Dazu kann die Messung der Akkuspannung herangezogen werden, um die Drehzahl bei sinkender Akkuspannung entsprechend zu regeln. Dies ist ein sehr einfach jedoch ein sehr ungenauer Weg die sinkende Akkuspannung zu kompensieren. Die bessere Variante ist es, den Drehzahlsensor für eine genaue Drehzahlregelung zu benutzen. Über die Drehzahlmessung kann auch eine Wegmessung realisiert werden.

#### **9-Achsen Bewegungssensor**

Mit diesem Sensor kann die Navigationsaufgabe des Fahrzeugs verbessert und erweitert werden. Dieser Sensor wird innerhalb der Laborübung nicht behandelt, darf aber beim Rennen benutzt werden.

### 11.3. Aktorik

#### Lenkservo / ESC

Durch die Ansteuerung mittels PWM, kann es bei momentanen Änderungen zu Totzeiten kommen, im schlimmsten Fall  $1/f_{\text{PWM}}$ . Speziell bei der Lenkung spielt dies eine Rolle, da diese nur in endlicher Zeit von einer auf die andere Position wechseln kann, z.B.: von 0 auf 60° ca. 0,2s. Die Dauer beim ESC beträgt ca. eine Periode des PWM Signals zuzüglich der Zeitverzögerung bei der Beschleunigung des Fahrzeugs.

### 11.4. Regelalgorithmen

#### PID Regler

Für die Regelung des Fahrzeugs, sowohl Lenkung als auch Geschwindigkeit, kann ein einfacher PID Regelalgorithmus benutzt werden. Unter folgendem Link finden sie eine kurze Einführung zum Thema Regler, <http://rn-wissen.de/wiki/index.php/Regelungstechnik>

Bei mathematischen Regelalgorithmen sollte immer drauf geachtet werden, dass diese in Integer und nicht Floatarithmetik gerechnet werden sollten, da der MSP430 keine Floating Point Arithmetic Unit besitzt, und somit diese Rechnung sehr viel Rechenzeit benötigen, vgl. 2.2 Rechendauer Performance Test.

#### Statemachine

Mit einer Statemachine werden verschiedene States durchlaufen abhängig davon welche Ereignisse eintreten.

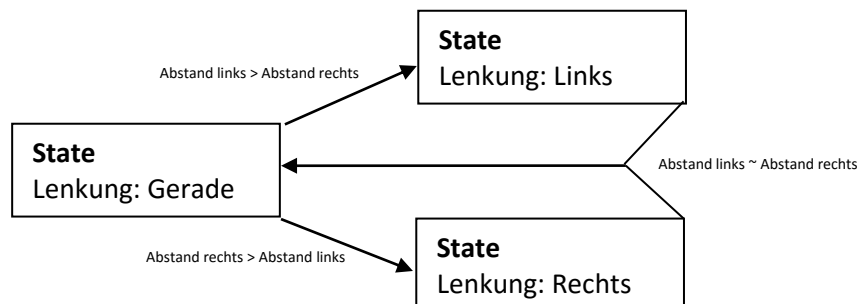


Abbildung 6: Flow Chart Statemachine.

Die Sensorwerte sollten bei Statemachines immer als Bereich angegeben werden, da diese einer gewissen Ungenauigkeit unterliegen.

## 11.5. Fahralgorithmen

Jeder der angeführten Fahralgorithmen kann mit jedem Regelalgorithmus umgesetzt werden.

### **Fahren entlang der Bande**

Eine einfache Möglichkeit die Strecke zu absolvieren ist es das Fahrzeug entlang einer Bande mit konstantem Abstand fahren zu lassen. Hierzu wird im einfachsten Fall nur ein Sensor benötigt. Weiters spielen die weitläufigen Kurven keine Rolle, da der benutzte Sensor immer die Banden im Empfangsbereich hat. Dieser Algorithmus kann durch den Sensor der gerade nach vorne misst, verbessert werden.

### **Fahren in der Mitte**

Bei diesem Algorithmus wird der Abstand nach links und rechts gemessen und das Fahrzeug soll sich mithilfe einer Regelung in der Mitte der Strecke halten. Für die weitläufigen Kurven, wo die Sensoren außerhalb des Messbereichs sind, muss ein alternativer Algorithmus verwendet werden.

### **TIPP: Testen des Fahralgorithmus**

Zu Beginn ist es sehr hilfreich, wenn keine Geschwindigkeitsregelung implementiert wird, und das Fahrzeug mit konstant langsamer Geschwindigkeit fährt.