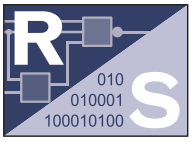
Implementierung eines Abstandsmessers auf Basis des SpartanMC

Rechnersystem 2 Praktikum von Tran Minh Quang und El horchi Amal
Tag der Einreichung: 27. Januar 2021

1. Gutachten: Prof. Dr.-Ing. Christian Hochberger
2. Gutachten: M.Sc.Johanna Rohde
3. Gutachten: David Volz
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT



Fachbereich Elektrotechnik
und Informationstechnik
Institut für Datentechnik
FG Rechnersysteme

Inhaltsverzeichnis

1 Einleitung

In diesem Projekt haben wir mit Hilfe von SpartanMC SoC Kits neben der Ultraschallsensoreinheit ein Distanzmesssystem entwickelt, dieser Ultraschallsensor misst einen Abstand und zeigt ihn dann auf einem OLED-Display an. Der Ultraschallsensor SRF02 muss sich im I2C-Modus befinden und das Display sollte im 3-Draht(Wire)-SPI-Modus betrieben werden. Das Projekt besteht auch aus einem C-Programm, das den Betrieb des Mikrocontrollers steuert.

Der erste Schritt ist, das OLED-Display über SPI zu bedienen. Ziel ist es, mit dem SpartanMC eine „Hello World“ auf dem Display zu erstellen. Im zweiten Schritt soll der Ultraschallsensor SRF02 in Betrieb genommen werden. Dies muss im I2C-Modus sein. Schließlich bringen wir die beiden Schritte, die Entfernungsmessung mit dem Sensor zusammen und zeigen die gemessene Entfernung in Zentimetern auf der OLED- Display an.

2 Systemarchitektur und Konfiguration mit Jconfig

Jconfig ist ein benutzerfreundliches Software-Tool, das für die Konfiguration von Spartan MC-Systemen verwendet wird. In diesem Teil stellen wir unsere Hardwarekonfiguration des SoC vor, die mit JConfig erstellt wurde. Zuerst wird es keine Interrupts verwendet. Danach wird I2C Interrupt und I2C Timer benuzt, um Takte der I2C Schnittstelle zu sparen und damit die Leistung des Systems zu optimieren.

2.1 SoC mit Interrupt

Das SoC auf Jconfig besteht aus:

- SpartanMC Core
- Xilinx DCM Clock
- SpartanMC Local Memory
- SPI Master
- I2C Master
- Output Port
- Intr_ctrl
- Timer

Das Abbildung 2.1 stellt das SoC dar:

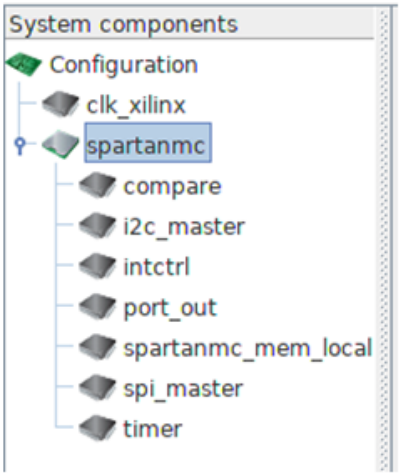


Abbildung 2.1: *Systemarchitektur mit Interrupts*

Wir haben den ausgewählten IP-Core in Jconfig konfiguriert, indem wir die Parameter und Verbindungen festgelegt haben, die den Anforderungen unserer Implementierung entsprechen.

Die folgenden Abbildungen zeigen ein Beispiel für die Konfiguration von Port_Out und Spi_master:

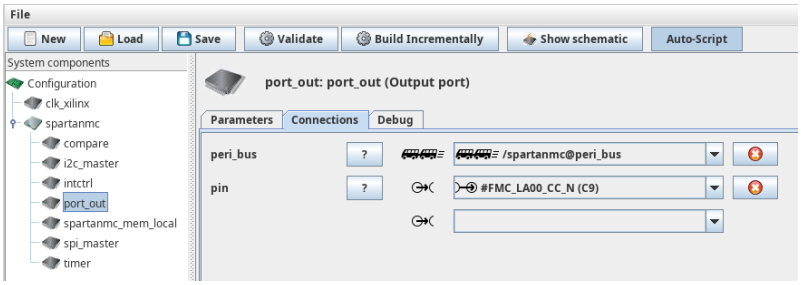


Abbildung 2.2: *Konfigurieren der Verbindungen von Port_Out*

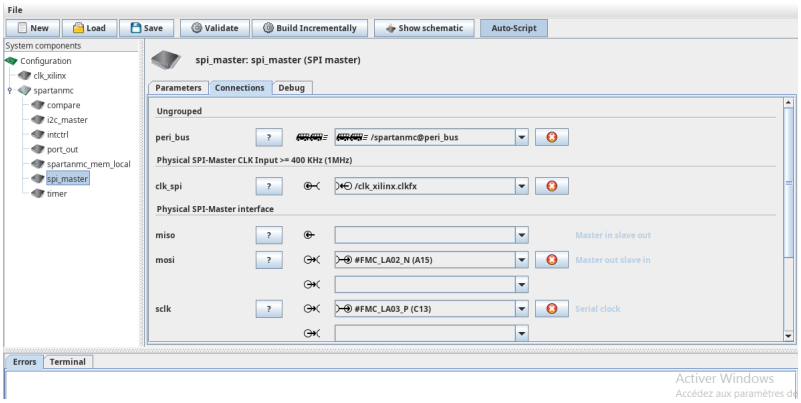


Abbildung 2.3: *Konfigurieren der Verbindungen von Spi_master*

3 Abstand Messung Implementierung

Um die Daten vom Sensor zu lesen, haben wir zwei Methoden implementiert. Eine besteht darin, einen Timer zu verwenden, um 65 ms zu warten. Die zweite Möglichkeit besteht darin, das Softwareregister kontinuierlich zu überprüfen. Wenn sich dieses Register von 0 unterscheidet, werden Daten abgerufen. Jede Methode hat ihre eigenen Vor- und Nachteile. Wir werden diese Punkte in den folgenden Abschnitten ausführlicher behandeln.

3.1 Treiber Bibliotheke

In unserem Programm haben wir Bibliotheken für I2C und SRF02 erstellt. Dadurch können wir die Funktionen einfach wieder anrufen, wenn wir sie brauchen. In I2C Bibliothek besteht aus folgenden Funktionen, die im Projekt benutzt werden, um mit SRF02 zu kommunizieren:

```
0
//Enable I2C peripheral and I2C interrupt.
2 void i2c_peri_enable(i2c_master_regs_t *i2c_master);

4 //Read data from selected registers.
void i2c_peri_read(i2c_master_regs_t *i2c_master, char slave_addr, char
    byte_to_read);
6
//Write data to selected register.
8 void i2c_peri_write(i2c_master_regs_t *i2c_master, char slave_addr, char
    byte_to_write, char *data, char stop);
```

Quelltext 3.1: *I2C driver*

In SRF02 Bibliothek besteht aus folgenden Funktionen, die im Projekt benutzt werden, um mit SRF02 zu arbeiten:

```
0 // Trigger a new ranging process.
void srf02_trigger( i2c_master_regs_t *i2c_master, char slave_addr,
    srf02_mode_t mode);
2
// Send the first part of read command, which indicate the registers should
     be read.
```

```
4 void srf02_register_set(i2c_master_regs_t *i2c_master, char slave_addr,
    char reg);

6 // Read data from slave.
  void srf02_read(i2c_master_regs_t *i2c_master, char slave_addr,int
    byte_to_read);
8

10 // Take received data from I2C buffer.
  void srf02_data_update(i2c_master_regs_t *i2c_master);
```

Quelltext 3.2: *srf05 driver*

Die oben genannten Funktionen reichen aus, um mit Sensoren im Timer-Modus zu arbeiten. Bei Firmware Register Polling werden zusätzlich die folgenden Funktion benötigt:

```
0 // Check if the data get back from firmware register is not equal to 255.
  // If it is equal to 255, send the request to read firmware register again
2 // If it is not equal, senf the request to read the data.
  void srf02_firmware_read(i2c_master_regs_t *i2c_master, char slave_addr);
```

Quelltext 3.3: *srf05 driver*

3.2 Abstand Messung mit Firmware-Version Polling

Für diese Methode verwenden wir den I2C-Interrupt für die Sensorkommunikation. Nach dem Senden des Befehls zum Starten der Entfernungsmessung zum Sensor lesen wir die Daten jedoch nicht sofort, sondern überprüfen das Firmware Register. Sobald das Lesen dieses Registers einen anderen Wert als 0 zurückgibt, hat der Sensor die Entfernungsmessung abgeschlossen. Wir holen die Daten ab.

Die Vorgehensweise der Messung kann in dem Diagramm 3.1 zusammengefasst:

Wir haben eine globale Variable „state“, um der letzte Stand abzuspeichern. Die Bedeutung der Stände im Diagramm wird wie folgt erklärt:

- TRIGGER: In diesem Stand wird der Befehl, der neue Messung anfangen, geschickt. Wenn die Übertragung schon erfolgreich durchgefürt wird, springt das Program in den Stand FIRWARE REGISTER SET.
- FIRWARE REGISTER SET: In diesem Stand wird der erste Teil des Lesen-Befehls, der zeigt, dass Register 0 abgelesen werden soll, geschickt.
- FIRMWARE READ: In diesem Stand wird der zweite Teil des Lesen-Befehls geschickt und prüft es, ob sich die Firmware-Version von 0 unterscheidet.

- SENSOR REGISTER SET: In diesem Stand wird der erste Teil des Lesen-Befehls, der zeigt, dass Register 2 abgelesen werden soll, geschickt.
- READING: In diesem Stand wird der zweite Teil des Lesen-Befehls geschickt.
- DATA UPDATE: In diesem Stand wird die Daten geholt.

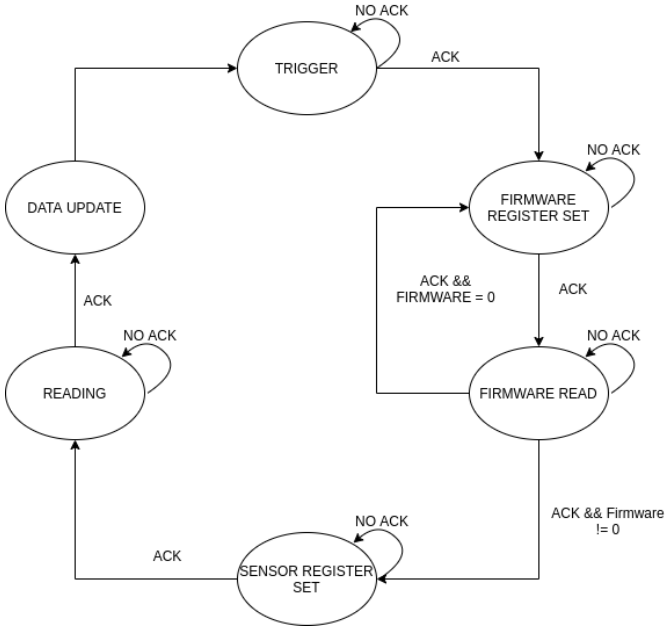


Abbildung 3.1: *State Machine mit Firmware Register Polling*

Wie verhält sich die Kommunikation zwischen dem Sensor und dem Prozessor wird in der Abbildung 3.2 gezeigt:

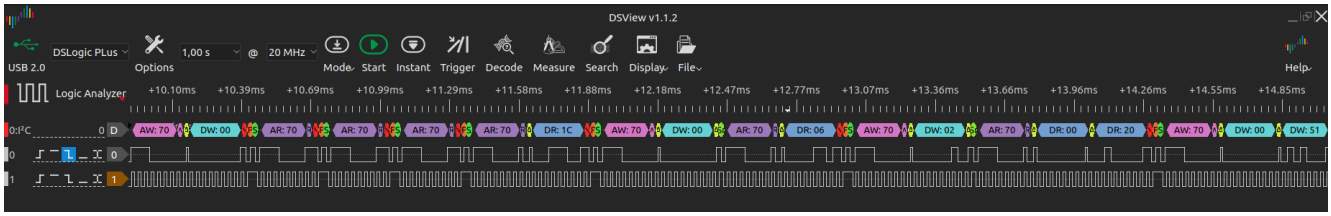


Abbildung 3.2: *Messung mit Firmware-Version Polling*

Diese Methode hat folgende Vor- und Nachteile:

- Vorteil: Die Ergebnisse können möglichst schnell bekommen werden.
- Nachteil: Mehrere Takte werden verschwendet, um die Firmware Version zu lesen, insbesonder für den Fall, dass die Entfernung groß ist.

Wenn die Anwendung eine schnelle Menge an Sensorinformationen erfordert, sollten wir diese Methode verwenden. Wenn dies jedoch nicht so wichtig ist und die CPU viele Aufgaben zu

erledigen hat, können wir einen zusätzlichen Timer-Interrupt verwenden, um die Takte maximal zu sparen.

3.3 Abstand Messung mit Timer Interrupt

In dieser Methode werden wir die maximale Messdauer abwarten. Dazu benutzen wir noch einen Timer-Capture-Interrupt, der genau 65ms dauert. Das bedeutet, dass wir 15 Messungen pro Sekunden durchführen können und das ist für viele Anwendungen schon ausreichen. Die Anwendung einer Timer-Capture-Interrupt ist zwar komplizierter und langsamer, aber wir können die Takte sowie die Energie optimal einsparen.

Die Vorgehensweise der Messung mit Interrupt kann in dem folgenden Diagramm zusammengefasst:

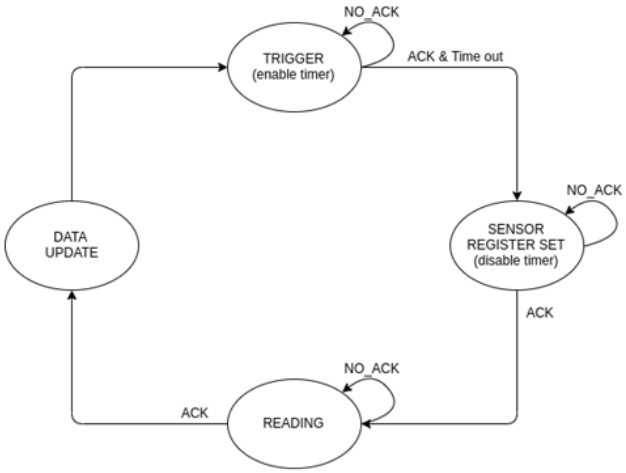


Abbildung 3.3: *State Machine der Timer Interrupt*

Wir haben eine globale Variable „state“, um der letzte Stand abzuspeichern. Die Bedeutung der Stände im Diagramm wird wie folgt erklärt:

- TRIGGER: In diesem Stand wird der Befehl, der neue Messung anfangen, geschickt. Wenn die Übertragung schon erfolgreich durchgefürt wird, fängt der Timer an zu zahlen. Nach der Abwartungszeit springt das Program in den Stand SENSOR REGISTER SET.
- SENSOR REGISTER SET: In diesem Stand wird der erste Teil des Lesen-Befehls, der zeigt, welche Register abgelesen werden soll, geschickt.
- READING: In diesem Stand wird der zweite Teil des Lesen-Befehls geschickt.
- DATA UPDATE: In diesem Stand wird die Daten geholt.

Wie verhält sich die Kommunikation zwischen dem Sensor und dem Prozessor wird in den folgenden Abbildungen gezeigt:

Das erste Bild stellt die Übertragung im TRIGGER-Stand dar: Danach fängt der Timer, der 65ms

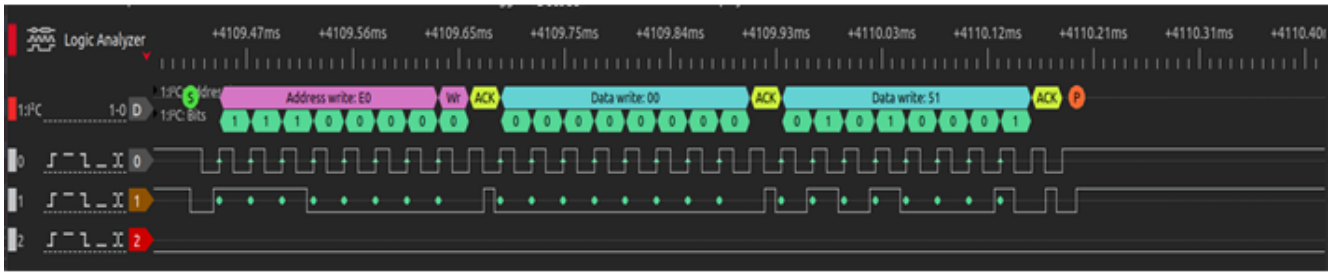


Abbildung 3.4: *Messung des Sensors mit Interrupts*

dauert, an zu zahlen:

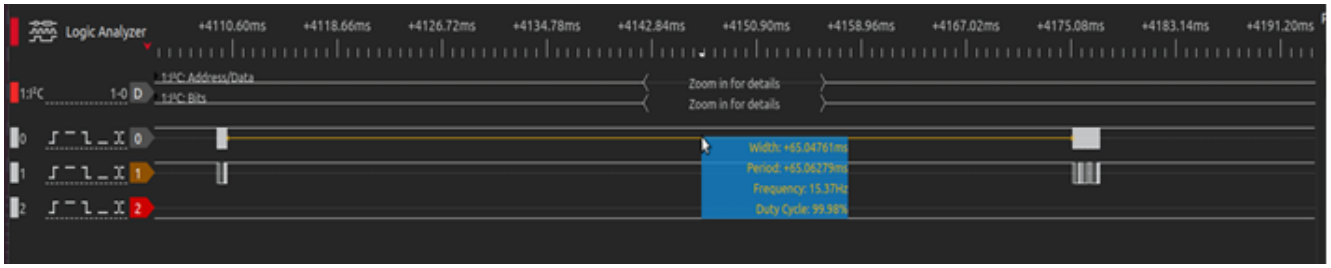


Abbildung 3.5: *Messdauer abwarten*

Nach der Abwartungszeit wird die Daten abgelesen und neue Messung wird angefangen:

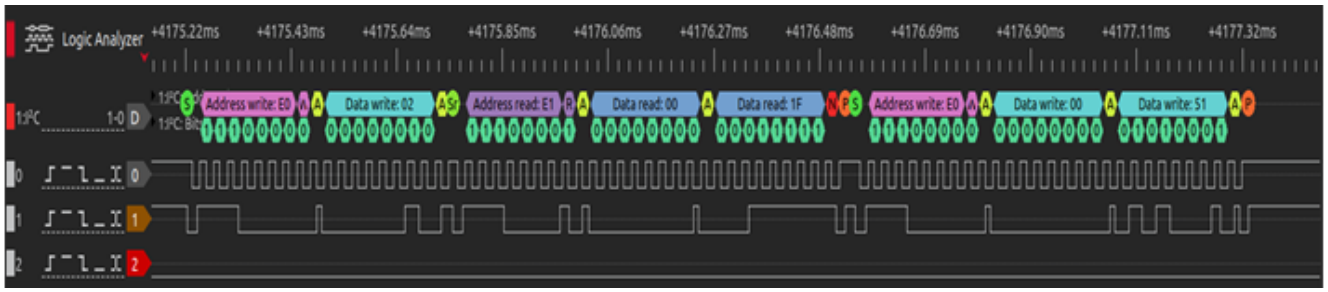


Abbildung 3.6: *Daten ablesen mit Interrupt*

3.4 Data Filter

Da die erhaltenen Werte ab und zu nicht stabil sind, wurde ein zusätzliche Median Filter implementiert. Die Eingabe dieser Filter ist die aktuellen Messdaten des Sensors, wird sie mit den

vorherigen Daten in einer Reihe speichert, sortiert und gibt der Filter den Wert an der mittleren Position zurück Wie viele Messdaten sollen gespeichert werden, ist mit der Variabel FILTER_SIZE in der „srf02.c“ Datei definiert. Die Wert von FILTER_SIZE soll nicht zu großgewählt werden, weil die Daten vom Sensor nicht so stark schwanken. Außerdem wenn wir den Wert zu groß wählen, führt dies zu einer Verzögerung des Lesedatenprozesses.

3.5 Entfernungsanzeige mit OLED

In dieser Aufgabe wurden zuerst vier SPI Treiber-Funktionen implementiert:

- void spi_peri_enable(): Control Registers einstellen
- void spi_peri_select(): Slave selektieren
- void spi_peri_deselect (): Slave deselektieren
- void spi_peri_write (): Daten oder Befehl schreiben

Danach, wurden zwei OLED-Funktionen implementiert, die die spi_peri_write () Funktion nutzen:

- void oled_Command_25664(): Befehl schreiben
- void oled_Data_25664 (): Daten schreiben

Zu der Anzeige der Entfernung auf dem Bildschirm wird die Funktion „Show_String_25664“ aufruft.

Die Vorgehensweise der Entfernungsanzeige kann in dem folgenden Diagramm zusammengefasst:

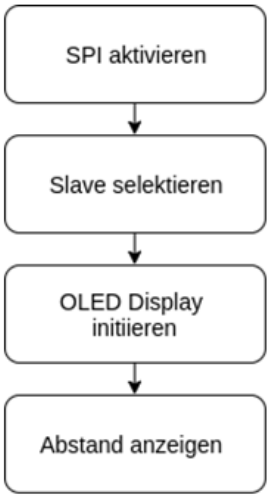


Abbildung 3.7: *Entfernungsanzeige mit OLED*

Die deselect Funktion muss nicht unbedingt verwendet werden, da wir nur ein Slave haben. Für diese Aufgabe, wird nur die Lösung mit Polling implementiert. Der Grund dafür ist, dass das Interrupt wird nur ausgelöst, wenn die Übertragung der Daten fertig ist und es wird Jedes Mal nur 9 Bits übertragt.Deswegen benötigen wir also ungefähr 70 bis 100 Takt pro Datenübertragung. Jedoch kann der Overhead der laufenden Unterbrechung langsamer als Polling sein, weil wir die Registerinhalte des Prozessors retten und restaurieren mussen. Darüber hinaus müssen wir noch den Inhalt des ISR durchfuhren. Das ist der Grund, warrum brauchen wir kein Unterbrechung für SPI.