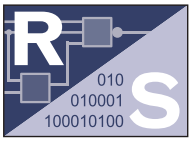
Implementierung eines Abstandsmessers auf Basis des SpartanMC

Versuchbericht von Yang Yin 2577159, Yue Zhao 2847821
Tag der Einreichung: 4. Juni 2023

1. Gutachten: Prof. Dr.-Ing. Christian Hochberger
2. Gutachten: M.Sc. Ramon Wirsch
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT



Fachbereich Elektrotechnik und Informationstechnik
Institut für Datentechnik
FG Rechnersysteme

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Yang Yin 2577159, Yue Zhao 2847821, die vorliegende Versuchbericht ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 4. Juni 2023

M. Mustermann

Inhaltsverzeichnis

1 Einleitung

Der SpartanMC ist ein speziell für FPGAs entwickelter Prozessor. Mit seiner ungewöhnlichen Befehls- und Datenbreite von 18 Bit nutzt er die Strukturen heutiger FPGAs optimal aus. Um den SpartanMC herum wurde in der Vergangenheit ein System-on-Chip Kit entwickelt, das eine Sammlung von Peripheriekomponenten und verschiedenen Softwarewerkzeugen bietet. Das verwendete 3-Schichten-Modell für SoCs besteht aus der physikalischen Schicht (PHY), der Soft-Core-Schicht (HDL) und der Firmware-Schicht (SW).
Die PHY umfasst die sichtbaren Hardwarekomponenten wie den Ultraschallsensor, das FPGA-Board und das OLED-Display. In der PHY wird das FPGA-Board ausgewählt und angeschlossen. In diesem Experiment verwenden wir das SP601 FPGA-Board.
Der Soft-Core und die Peripheriekomponenten werden in HDL mit Jconfig konfiguriert. UART, SPI und I2C werden in diesem Versuch verwendet. In der SW-Schicht sind verschiedene C-Dateien enthalten, und die Firmware kann entwickelt werden.
Im Rahmen dieser Übung soll mit Hilfe des SpartanMC SoC Kits ein Abstandsmesser erstellt werden. Dieser misst den Abstand mithilfe eines Ultraschallsensors und zeigt ihn dann auf einem OLED-Display an.

1.1 Aufgeben

1) **OLED Display über SPI**
   Mit Hilfe des SpartanMC kann ein "Hello Worldäuf dem OLED Display anzeigen.
2) **Steuerung von Ultraschallsensor SRF02**
   Aufgabe ist es, eine Messung zu starten und das Ergebnis über die Konsole des PCs auszugeben.
3) **Implementierung des Entfernungsmessers**
   Basierend auf Aufgabenteil 1 wird das System umgebaut, um einen funktionierenden Abstandsmesser zu erstellen.

4) **Verbesserungen der Abstandsmessergenauigkeit**
   Gelegentlich kann es vorkommen, dass der Sensor ungenaue oder falsche Messwerte produziert. Um diese vor dem Anwender zu verbergen, soll eine Filterung eingebaut werden.

5) **Einsparung von Rechenleistung**
   Erfahrungsgemäß führt der naive Ansatz dazu, dass die Firmware viel Ausführungszeit darauf verwendet, auf ein bestimmtes Event zu warten. Um diese verschwendete Rechenleistung nutzbar zu machen, soll die Lösung durch Interrupts erweitert werden.

2 Konfiguration & Programme

Zuerst sollten wir Aufgabe 1 abschließen, indem wir das OLED-Display über SPI in Betrieb nehmen und mithilfe des SpartanMC "Hello Worldäuf dem Display anzeigen. Danach werden wir Aufgabe 2 auf der Grundlage von Aufgabe 1 abschließen, indem wir den SRF02-Ultraschallsensor im I2C-Modus betreiben. Die Abstandsmessung beginnt und die gemessenen Ergebnisse werden auf dem Display angezeigt. Dadurch werden wir Aufgabe 3 abschließen.

2.1 Aufbau der JConfig

Zuerst ist mit Hilfe des Systembuilders JConfig ein SoC aufzubauen, welches aus folgenden Modulen besteht:

- SpartanMC Core: Die Hardwarekomponenten, die im SoC verwendet werden sollen, müssen ausgewählt werden.
- Xilinx DCM Clock: Der Clock-Generator wird automatisch hinzugefügt.
- SpartanMC Local Memory:
- SPI Master: Das physische Interface des Displays wird mit Hilfe der Dokumentation des Aufsteck-Boards [fpga_pins] konfiguriert. Das Ziel ist es, den Clock Port, den Daten port und den Slave Select Port anzuschließen. Siehe Abbildung 2.1
- Output Port: Das Output muss an den Reset Port des Displays angeschlossen werden. Siehe Abbildung 2.2
- I2C: Das Interface des I2C Master wird mit Hilfe der Dokumentation des Aufsteck-Boards [fpga_pins] konfiguriert. Dabei werden der Clock Port und der Data Port des I2C-Masters angeschlossen. Siehe Abbildung 2.3
- Uart_light: Die UART-Schnittstelle dient dem Senden und Empfangen von Daten. Der UART wird gemäß dem Quick Start Guide [quickGuide] konfiguriert. Die Frequenz des UART sind glechen wie die des SpartanMC Core. Siehe Abbildung 2.4

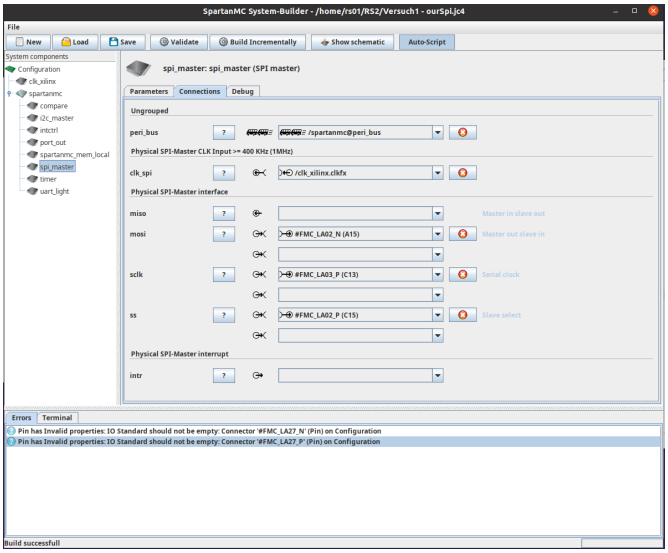


Abbildung 2.1: SPI Master

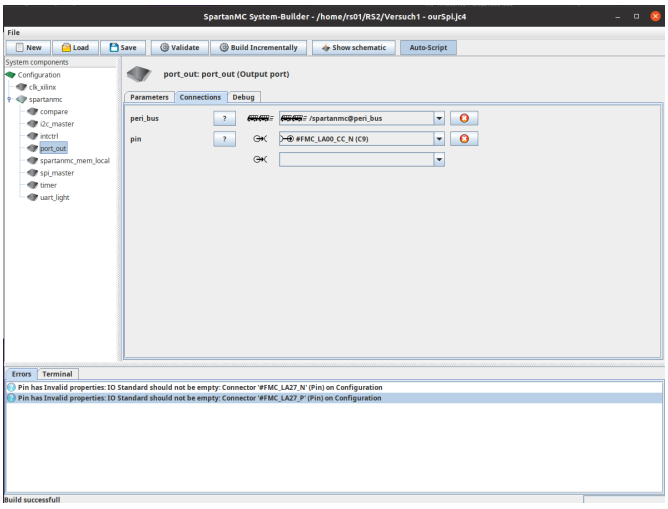


Abbildung 2.2: Output Port

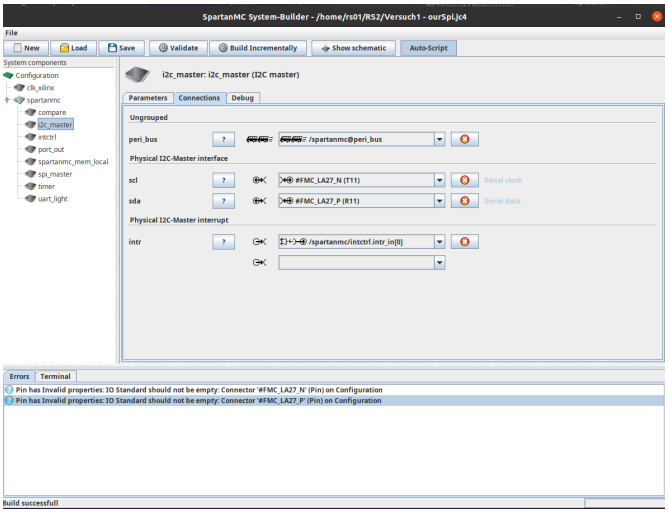


Abbildung 2.3: I2C

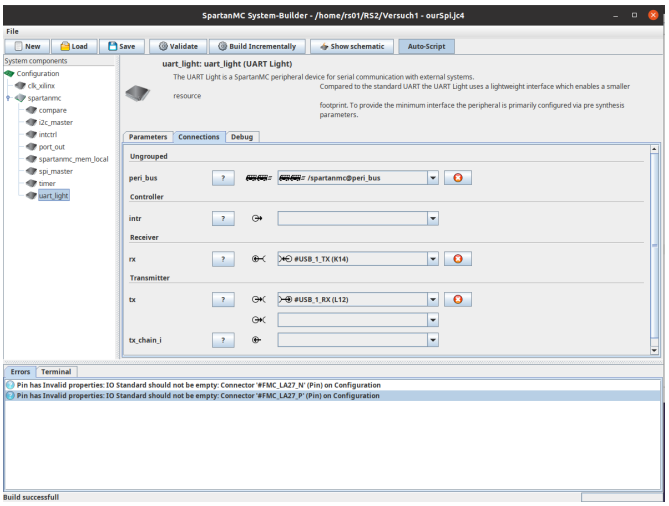


Abbildung 2.4: Uart_light

2.2 Programmierung

2.2.1 Betrieb von OLEDs

1) **Initialisierung**
   Um SPI zu initialisieren, müssen wir zuerst einige Parameter in das Controlregister der SPI-Peripherie des SpartanMC schreiben.

```
void spi_peri_enable(){
    SPI_MASTER.spi.spi_control &= ResetMask;
    SPI_MASTER.spi.spi_control |= SPI_MASTER_CTRL_EN;
    SPI_MASTER.spi.spi_control &= ~SPI_MASTER_CTRL_BITCNT;
    SPI_MASTER.spi.spi_control |= 0x00900U; // 9 bit SPI frame
    SPI_MASTER.spi.spi_control |= 0x02000U; // set the frequency of
       Display to 7.5 MHz
}
```

2) **Aktivierung**
   Bevor mit dem OLED kommuniziert werden kann, muss das Slave-Select-Signal im SPI-Controlregister aktiviert werden.

```
void spi_peri_select(){
    SPI_MASTER.spi.spi_control |= SPI_SLAVE_ADDRESS;
}
```

3) **Befehlschreiben**
   Das OLED-Display muss auch durch verschiedene Befehle initialisiert werden, die über den SPI-Bus empfangen werden. Das SPI_data_out-Register speichert die zu versendenden Daten.

```
void spi_peri_write(const unsigned char data){
    while(SPI_MASTER.spi.spi_status & SPI_MASTER_STAT_FILL);
    SPI_MASTER.spi.spi_data_out=data;
}
```

4) **Deaktivierung**
   Am Ende des Programms kann der SPI-Bus deaktiviert werden.

```
void spi_peri_deselect(){
    SPI_MASTER.spi.spi_control &= ~SPI_MASTER_CTRL_SLAVE;
}
```

2.2.2 Betrieb von Ultraschallsensor

1) Aktivierung
Wie bei SPI müssen wir auch die Controlregister von I2C vor der Verwendung konfigurieren. Das Enable-Bit wird gesetzt. Den Prescaler setzen wir gemäß folgender Formel:

$$PRESCALER = sys\_clk/(5 * desired\_SCL) \quad - \quad 1 \tag{2.1}$$

Davon ist Systemtakt 60MHz und erwünschte Takte für I2C ist 30KHz. Dann ist es einfach zu berechnen: P RESCALAR = 399.

```
void i2c_peri_enable(){
    I2C_MASTER.ctrl &= ResetMask;
    I2C_MASTER.ctrl |= I2C_CTRL_EN | I2C_PRESCALER;
    I2C_MASTER.ctrl |= I2C_CTRL_INTR_EN;
}
```

2) Befehlschreiben
Wir müssen die Adresse von Slave und Zielregisternummer an die I2C-Peripherie senden. Anschließend können die Befehle gesendet werden.

```
void i2c_peri_write(const unsigned char data){
    I2C_MASTER.data[0]= I2C_SLAVE_ADDRESS<<1; //set the last bit to 0,
       indicating for writing
    I2C_MASTER.data[1] = 0; // set registerpointer to #0
    I2C_MASTER.data[2] = data;
    I2C_MASTER.cmd = I2C_CMD_STA | (3<<3) | I2C_CMD_STO;

    //while (I2C_MASTER.stat & I2C_STA_TIP);
    //while(I2C_MASTER.stat & I2C_STA_NO_ACK);
}
```

3) Das Senden des Lesebefehls
Die Register zum Speichern der Entfernung sind das Zweite und Dritte. Vor dem Lesen muss dem Slave mitgeteilt werden, dass Register 2 das Zielregister ist, das gelesen werden soll.

```
void sendReadCommand(){
    I2C_MASTER.data[0]= (I2C_SLAVE_ADDRESS<<1); //set the last bit to
       0,indicating for writing
    I2C_MASTER.data[1] = 2; // set registerpointer to #2
    I2C_MASTER.cmd = I2C_CMD_STA | (2<<3) | I2C_CMD_STO;
    //while (I2C_MASTER.stat & I2C_STA_TIP);
    //while(I2C_MASTER.stat & I2C_STA_NO_ACK);
}
```

4) Lesen
Um die Daten zu lesen, muss die Slaveadresse um 1 erhöht werden. Das RD-Bit im I2C-Commandregister muss ebenfalls gesetzt werden.

```
    void i2c_peri_read(){
        I2C_MASTER.data[0]= (I2C_SLAVE_ADDRESS<<1)+1;//set the last bit to
            0,indicating for reading
        I2C_MASTER.cmd = I2C_CMD_STA | (3<<3) | I2C_CMD_RD;// read 2 bytes
            from sensor
        //while(I2C_MASTER.stat & I2C_STA_TIP);      //wait for not busy
    }
```

2.3 Filter-Funktion

Die vom Ultraschall gemessenen Daten sind gelegentlich ungenau oder falsch. Um dieses Problem zu verbessern, verwenden wir die Methode der Medianfilterung. Dafür verwenden wir einen Filter, der darauf basiert, den Median zu finden. Wir speichern die aufeinanderfolgenden Messwerte in einem Array und sortieren es. Dann wird der Median des Arrays als unser gewünschter Wert ausgegeben.

```
void midPassFilterInit(){
    midPassFilter.result =0;
    for(unsigned int i =0;i < ( sizeof(midPassFilter.val) / sizeof(unsigned
        int) ) ; i++){
        midPassFilter.val[i] = 0;
    }
}
void startMidPassFilter(){
    int length = sizeof(midPassFilter.val)/sizeof (unsigned int);
    midPassFilter.val[0] = sensor.result;
    for( int i = 0; i<length-1; i++){
        midPassFilter.val[length-1-i] = midPassFilter.val[length-2-i];
    }
    quicksort(0,4);
    midPassFilter.result = midPassFilter.val[2];
}
void quicksort(int leftPosi, int rightPosi) {
    if (leftPosi < rightPosi) {
        int pivot = midPassFilter.val[(leftPosi + rightPosi) / 2];
        int left = leftPosi;
        int right = rightPosi;
        while (left <= right) {
            while ((int)midPassFilter.val[left] < pivot) left = left + 1;
            while ((int)midPassFilter.val[right] > pivot) right = right -
                1;
            if (left <= right) {
                int temp = midPassFilter.val[left];
                midPassFilter.val[left] = midPassFilter.val[right];
```

```
                midPassFilter.val[right] = temp;
                left = left + 1;
                right = right - 1;
            }
        }
        quicksort(leftPosi, right);
        quicksort(left, rightPosi);
    }
}
```

3 Implementierung

3.1 Polling und Sleepfuntion

3.1.1 Ausgabe auf OLED Display

Wir verwenden die Sleepfuntion, um das Ende der Messung zu erkennen. Danach wird die gemessene Entfernung an den OLED-Display gesendet und schließlich angezeigt. Siehe Abbildung.3.1

Abbildung 3.1: Messergebnisse auf dem Display

3.1.2 Sleepfuntion

```
void version1(){
    startOled();
    midPassFilterInit();
    i2c_peri_enable();
    while(1){
        startSensor();
        sleep(0.08);
        sendReadCommand();
        i2c_peri_read();
        readInCm();
        startMidPassFilter();
        showResult();
    }
    spi_peri_deselect();
}
```

3.2 Interrupt und Statemachine

Die Sleepfunktion bedeutet, dass die Firmware viel Ausführungszeit darauf verwendet, auf ein bestimmtes Event zu warten. Um diese verschwendete Rechenleistung nutzbar zu machen, werden Interrupts eingeführt.

3.2.1 Interrupt

Interrupts werden eingeführt. Die konfiguration in JConfig ist wie folgt:

- Interrupt controller: Siehe Abbildung 3.2
- Compare: Siehe Abbildung 3.3
- Timer: Siehe Abbildung 3.4

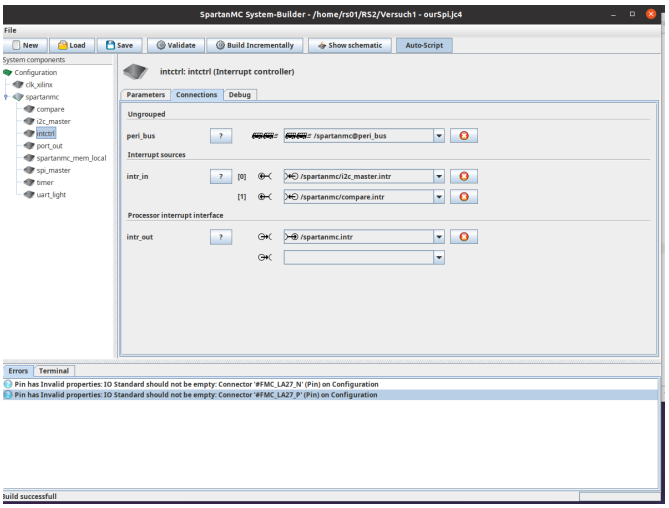


Abbildung 3.2: Interrupt controlle

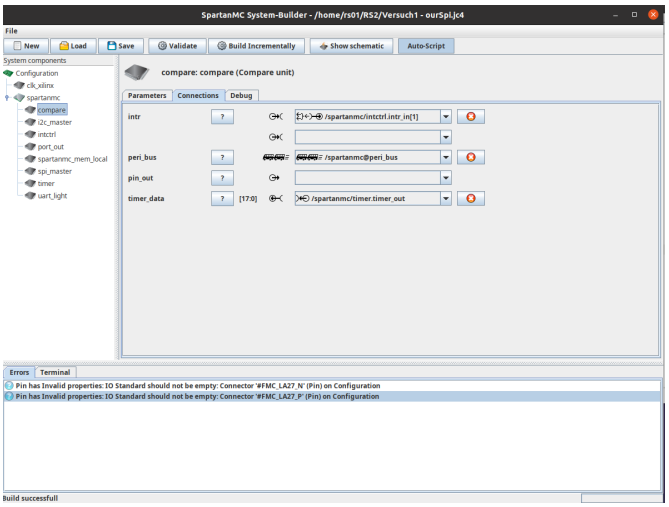


Abbildung 3.3: Compare

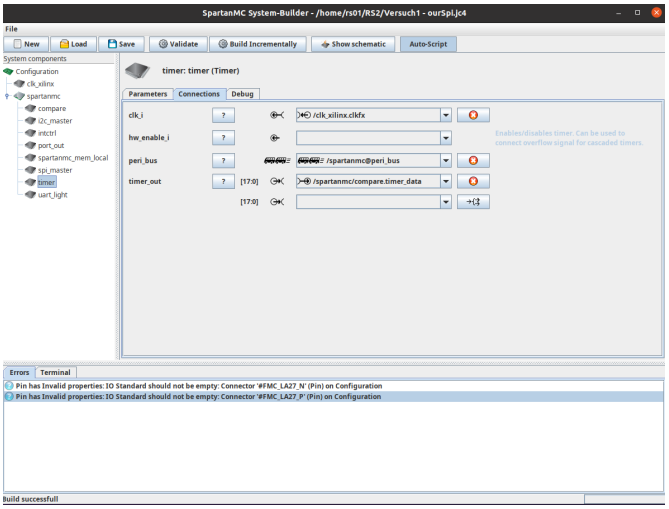


Abbildung 3.4: Timer

3.2.2 Statemachine

1) **Statemachine ist in der Abbildung3.5 dargestellt:**

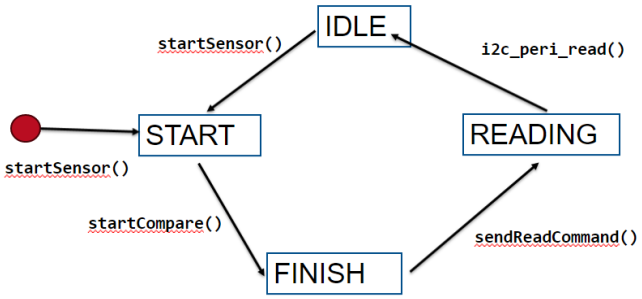


Abbildung 3.5: Statemachine

2) **Das Programm von Statemachine ist wie folgt:**

```
ISR(0)(){
printf("get in ISR 0 with flag %u\n",flag); // man braucht die funktion
    ISR zu verzögern
switch(flag){
case IDLE:
flag = START;
I2C_MASTER.data[0]=0;//stop the ISR
readInCm();
startSensor();
break;

case START:
flag = FINISH;
I2C_MASTER.data[0]=0;
startCompare();
break;

case READING:
flag = IDLE;
I2C_MASTER.data[0]=0;
i2c_peri_read();

break;

default:
flag = IDLE;
I2C_MASTER.data[0]=0;
```

```
printf("Invalid Parameter got ! \n");
break;
}
}

/*
 * the ISR of Timer interrupt source
 * */
ISR(1)(){
//printf("get in ISR 1 with flag %u\n",flag);
if(flag == FINISH){
flag =READING;
stopCompare();
sendReadCommand();
}
else{
printf("wrong Flag\n");
}
}
```

3.2.3 Ergebnisse unter Verwendung von Interrupt und Statemachine

Der Messwert kann auf dem Display angezeigt werden. Aber im Vergleich zur Methode mit Sleepfunktion wird die Entfernungswert auf dem OLED-Display nicht deutlich angezeigt.

4 Benutzungshinweise

Version 1 wird im Sleep-Modus ausgeführt, während Version 2 im Interrupt-Modus läuft.
Wenn Sie unser Programm im Sleep-Modus testen möchten, kommentieren Sie bitte Version 2 aus und entfernen Sie die Kommentare der While-Schleife im Treiber.
Wenn Sie unser Programm im Interrupt-Modus testen möchten, kommentieren Sie bitte Version 1 und die While-Schleife im Treiber aus.

Außerdem man muss beim Eintritt in IRS(0), die UART funktion printf() aufrufen. Sonst wird der Sensor nicht richtig gestarted. Es liegt daran, dass die IRS(0) zu schnell ist und ein Sychronisierensproblem auftritt. Die UART Funktion liefert uns die Möglichkeit, die aktuelle Zustände zu prüfen. Sie hat außerdem die ISR(0) verzögert, damit die Zustandsmaschine richtig laufen kann.

5 Evaluation

- **Vor- und Nachteile für Sleep-Funktion und Interrupt und Statemachine**
  Die Abstandsmessung ist mit Hilfe der Sleep-Funktion einfacher umzusetzen und die Ausgabe auf dem OLED-Display ist klarer. Allerdings führt die Zeitverzögerung zu einer Verschwendung der Rechenleistung des Controllers. Durch die Einführung von Interrupts muss das Programm nicht mehr auf die Bereitschaft des Slaves warten. Der I2C-Master kann einen Interrupt auslösen, sobald der aktuelle Sende-/Lesebefehl abgeschlossen ist. Die gelesenen Daten müssen nur im Interrupt verarbeitet werden. Dadurch wird die Effizienz des Controllers verbessert.

- **Die Gründe für die Nichtanwendung von SPI-Interrupts**
  Der SPI-Master kann auch Interrupts auslösen, sobald eine Übertragung abgeschlossen ist. Jedoch ist die maximale Ultraschallfrequenz 40 kHz, während die Peripheral-Clock der Sys-Clk 60 MHz entspricht. Im Vergleich dazu beträgt der Display-Clock 10 MHz, und die System-Clock ist nur 4-mal höher als der SCLk des Displays. Aufgrund dieser Tatsache ergibt es keinen Sinn, SPI-Interrupts zu verwenden.

6 Zusammenfassung

In diesem Experiment haben wir gemäß der Aufgabenstellung erfolgreich alle Versuchsaufgaben abgeschlossen und basierend auf dem SpartanMC einen Abstandsmesser erstellt. Der Abstandsmesser verwendet einen Ultraschallsensor zur Messung der Distanz und zeigt die Ergebnisse auf einem OLED-Display an. Insgesamt haben wir das Display mithilfe von SPI und dem I2C-Ultraschallsensor zum Laufen gebracht. Zunächst haben wir die Implementierung des Abstandsmessers mit der Sleep-Funktion durchgeführt. Um die Genauigkeit des Entfernungsmessers zu verbessern, haben wir einen Medianfilter entwickelt, der dazu dient, einige Ausreißerwerte zu filtern. Die Ergebnisse der Entfernungsmessungen werden auf dem OLED-Display angezeigt.
Jedoch haben wir festgestellt, dass die Verwendung der Sleep-Funktion die Rechenleistung verschwendet. Aus diesem Grund entscheiden wir uns, I2C-Interrupts einzusetzen und die Sleep-Funktion durch einen Timer-Compare-Interrupt zu ersetzen. Dadurch wurde die Effizienz des Controllers verbessert.
Zum Abschluss möchten wir uns bei unserem Professor und unseren Betreuern für ihre Unterstützung bedanken.

Abkürzungsverzeichnis

FPGA Field Programmable Gate Array

SoC System on Chip

PHY Physikalische Schicht

HDL Soft-Core-Schicht

SW Firmware-Schicht

UART Universal Asynchronous Receiver Transmitter

SPI Serial Peripheral Interface Bus

I2C Inter-integrated Circuit

ISR Interrupt Service Routine

Abbildungsverzeichnis

Literatur

[1] *BUG Xilinx XDL Programm*. URL: http://forums.xilinx.com/t5/Spartan-Family-FPGAs/A-bug-for-Spartan-6/td-p/271026.

[2] *Executable and linkable Format*. URL: http://www.linux-kernel.de/appendix/ap05.pdf.

[3] *GNU - make*. URL: http://www.gnu.org/software/make/manual/make.html.

[4] *XDL Use Case Senarios*. URL: http://www.cs.indiana.edu/hmg/le/project-home/xilinx/ise_5.2/help/data/xdl/xdl-ucs-ext.html.

[5] *Xilinx Design Language*. URL: http://www.cs.indiana.edu/hmg/le/project-home/xilinx/ise_5.2/help/data/xdl/xdl.html.