

Praktikum Rechnerorganisation

Versuch 4: Stoppuhr

Lernziele

- Erstellen einer Peripheriekomponente
- Programmieren von Peripheriezugriffen
- Einbinden der erstellten Komponente in das Beispielrechner-System
- VHDL-Simulation des Systems
- Synthese und Test mit realer Hardware

Werkzeuge

- BASYS3-Board
- ModelSim (Intel FPGA Starter Edition)
- Vivado (Version 2019.2)
- Eclipse IDE for C/C++ Developers (Version 2020-12)
- GNU-Toolchain für MIPS

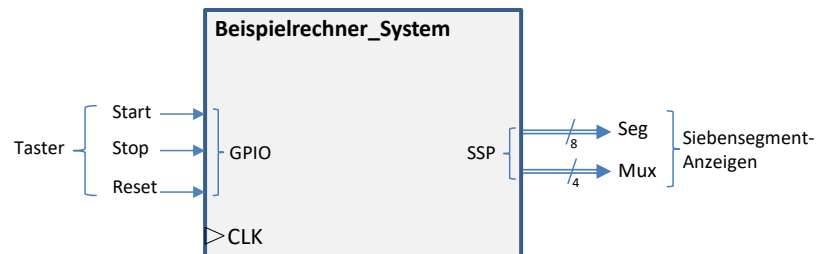
Vorbereitung (Vor dem Praktikumstermin)

- Wiederholen Sie den Abschnitt der Vorlesung über die Peripherie.
- Lesen Sie den Abschnitt Aufgabenstellung durch

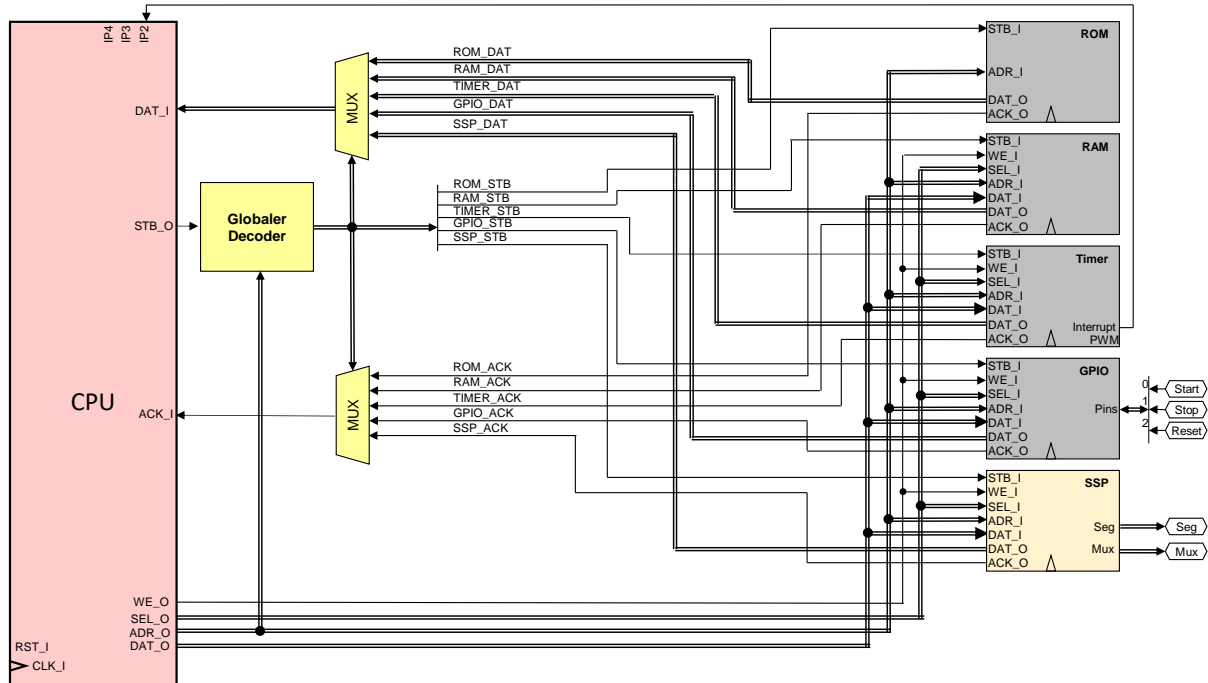
Systembeschreibung

In diesem Praktikumsversuch soll mit dem Beispielrechner und geeigneter Peripherie eine Stoppuhr realisiert werden. Die Steuerung erfolgt über Taster, die Ausgabe der Zeit über die Siebensegmentanzeigen auf dem BASYS3-Board.

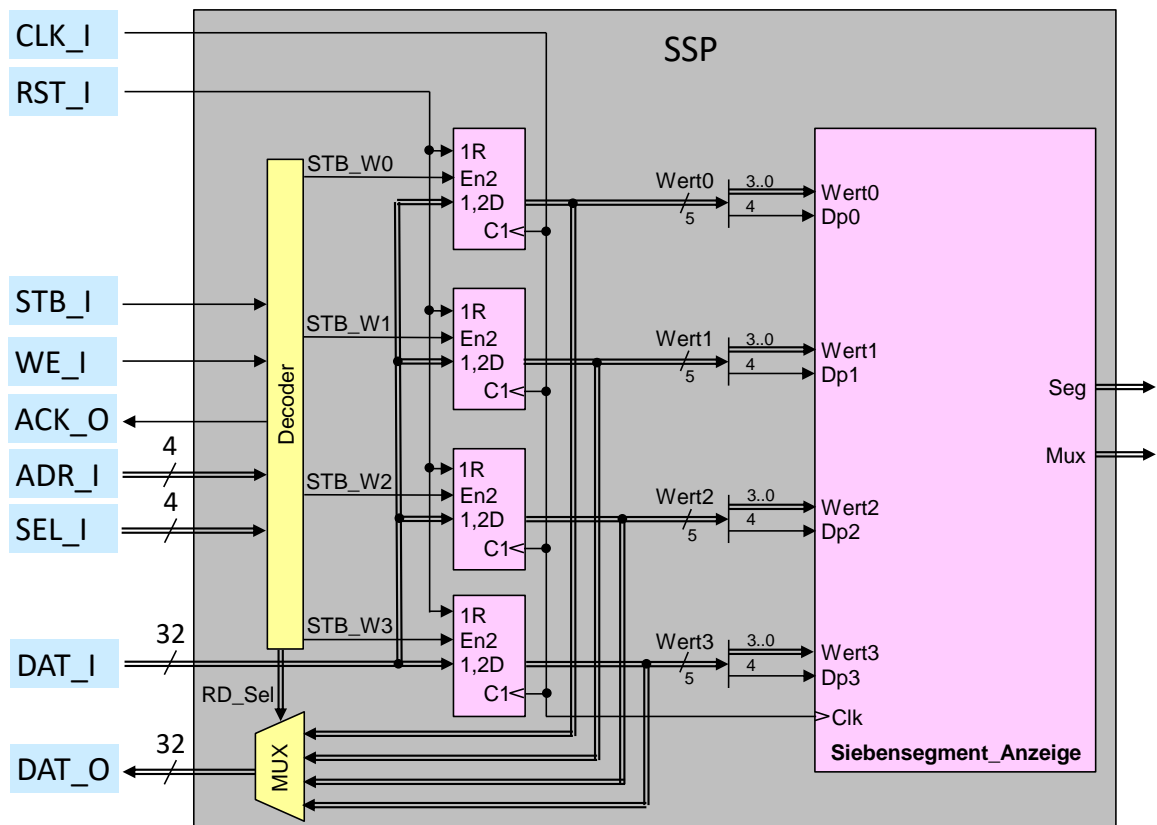
Folgende Übersicht zeigt den Aufbau:



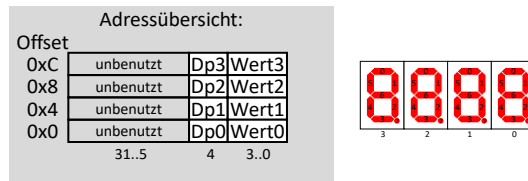
Ein vorgegebenes Beispielrechner-System wird um eine neue Peripheriekomponente *SSP* (Siebensegment-Port) ergänzt. Die folgende Abbildung zeigt das Blockdiagramm dieses Systems:



Die neue Komponente *SSP* wird in Aufgabe 1 dieses Versuchs erstellt. Ihr interner Aufbau wird in der folgenden Abbildung gezeigt:



Für jede Stelle der Anzeige ist ein rücklesbares Register vorgesehen, welches in seinen unteren 4 Bits (Bits 3..0) den darzustellenden Zahlenwert und in Bit 4 den Wert des zugehörigen Dezimalpunkts speichert. Schreibzugriffe auf die unbenutzten Bits 31..5 der Register werden ignoriert. Beim Lesen liefern diese Bits den Wert 0.



Software

Die Software des Systems führt zunächst eine Initialisierung aus. Dann beginnt die (endlose) Hauptschleife. In der Hauptschleife wird je nach Betriebszustand eine von zwei Schleifen ausgeführt:

- **WARTEN:** Es wird auf das Drücken des START- oder des RESET-Tasters gewartet. Durch Drücken des RESET -Tasters werden alle Stoppuhr-Werte (Minuten, Zehner, Einer, Zehntel) zu 0 gesetzt. Durch Drücken des START-Tasters wird diese Schleife verlassen, der Timer-Interrupt wird freigegeben und es beginnt die Ausführung der Schleife ZAEHLEN.
- **ZAEHLEN:** Die Stoppuhr wird weitergeschaltet. Zum Weiterschalten der Stoppuhr wird getestet, ob der Millisekunden-Wert *ms* durch die im Hintergrund laufende Timer-Funktion um mindestens 100 (entspricht einer Zehntelsekunde) hochgezählt wurde. In diesem Fall wird der Zehntelsekunden-Wert *Zehntel* inkrementiert und entsprechend 100 vom Millisekunden-Wert *ms* subtrahiert. Entsprechend werden die Stoppuhrwerte *Einer*, *Zehner* und *Minuten* auf Überlauf getestet und bei Bedarf die nächsthöheren Werte hochgezählt. Falls der STOP-Taster gedrückt ist, wird die Schleife verlassen, der Timer-Interrupt deaktiviert und die Schleife WARTEN wird wieder aktiv.

Folgende an die Sprache C angelehnte Softwarebeschreibung erläutert nochmals die Funktion der Stoppuhr (die am Rand markierten Programmteile müssen noch von Ihnen in Assembler übersetzt werden). Die Umsetzung des gezeigten Programms in Assemblerbefehle erfolgt manuell. Diese Umsetzung ist schon weitgehend durchgeführt und wird in der Datei [Stoppuhr.S](#) zur Verfügung gestellt.

```
// -----
// Peripherie-Definitionen
// -----
#define Timer_Basis          0x8000
#define Timer_Periode        0x0
#define Timer_Schwelle       0x4
#define Timer_Zaehlerstand   0x8
#define Timer_Kontroll       0xC
#define Timer_Status         0x10

#define Timer_IrEn           0

#define GPIO_Basis           0x8100
#define GPIO_Eingabe         0x0
#define GPIO_Ausgabe         0x4
#define GPIO_Richtung        0x8

#define SSP_Basis            0x8200
#define SSP_Wert0             0x0
#define SSP_Wert1            0x4
#define SSP_Wert2            0x8
#define SSP_Wert3            0xC

// -----
// Anwendungsspezifische Definitionen
// -----
// Spezifikation der GPIO-Pins
#define START                 0
#define STOP                  1
#define RESET                 2

# Konstanten fuer die Ausfuehrung in Hardware (Echtzeit)
#define ZAEHLER_PERIODE       59999 # Echtzeit (Ein Interrupt pro ms)
#define MS_PRO_INTERRUPT      1     # Echtzeit (1 ms pro Interrupt)

# Konstanten fuer die Ausfuehrung im VHDL-Simulator (Beschleunigung x100)
// #define ZAEHLER_PERIODE 29999    # Beschleunigt (2 Interrupts pro ms)
// #define MS_PRO_INTERRUPT 50      # Beschleunigt (50 ms pro Interrupt)
```

```

// -----
// Unterprogramme (aus Versuch 3 übernommen)
// -----

int bit_von(int Wert, int Bitnummer);
int lese_bit(int Adresswert, int Bitnummer);
void setze_bit(int Adresswert, int Bitnummer);
void loesche_bit(int Adresswert, int Bitnummer);
void schreibe_bitfeld(int Adresswert, int Wert, int Maske, int Shift);

// Globale Variable
volatile int ms; // Zaehler der Millisekunden

// Interrupt-Handler
// Diese muss irgendwie jede Millisekunde aufgerufen werden
void Timer_Handler() {
    ms = ms + MS_PRO_INTERRUPT;

    // Timer-Status abfragen, um Interrupt zu löschen
    *((int*)(Timer_Basis + Timer_Status));
}

// Unterprogramm zur Anzeige der Zeit
void Zeit_anzeigen(int Minuten, int Zehner, int Einer, int Zehntel) {
    *((int*)(SSP_Basis + SSP_Wert3)) = Minuten | 0x10; // Dezimalpunkt an;
    *((int*)(SSP_Basis + SSP_Wert2)) = Zehner;
    *((int*)(SSP_Basis + SSP_Wert1)) = Einer | 0x10; // Dezimalpunkt an;
    *((int*)(SSP_Basis + SSP_Wert0)) = Zehntel;
}

// Hauptprogramm
int main() {
    int Minuten = 0;
    int Zehner = 0;
    int Einer = 0;
    int Zehntel = 0;
    ms = 0;

    // Timer initialisieren
    schreibe_bitfeld(Timer_Basis + Timer_Periode, TAKTZYKLEN_PRO_MS, 0xffffffff, 0);

    // Interrupt IP2 irgendwie freigeben
    // Globalen Interrupt irgendwie freigeben

    Zeit_anzeigen(Minuten, Zehner, Einer, Zehntel);

    while(1) {

        // Schleife für Betriebszustand WARTEN
        while (1) {

            // Bei gedrücktem START-Taster Schleife WARTEN verlassen
            if (lese_bit(GPIO_Basis + GPIO_Eingabe, START) != 0) {
                break;
            }

            // Bei gedrücktem RESET-Taster alle Werte zurücksetzen
            if (lese_bit(GPIO_Basis + GPIO_Eingabe, RESET) != 0) {
                Minuten = 0;
                Zehner = 0;
                Einer = 0;
                Zehntel = 0;
                Zeit_anzeigen(Minuten, Zehner, Einer, Zehntel);
            }
        } // Ende der Schleife WARTEN

        // Uhr starten (Timer-Interrupt freigeben)
        ms = 0;
        setze_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn);
    }
}

```

```
// Schleife für Betriebszustand ZAEHLEN
while (1) {
    if (ms >= 100) {
        loesche_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn); // Timer-IR sperren
        ms = ms - 100;
        setze_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn); // Timer-IR freigeben
        Zehntel++;
        if (Zehntel == 10) { Einer++;   Zehntel = 0; }
        if (Einer == 10) { Zehner++;  Einer = 0; }
        if (Zehner == 6) { Minuten++; Zehner = 0; }
        if (Minuten == 10) {           Minuten = 0; }
        Zeit_anzeigen(Minuten, Zehner, Einer, Zehntel);
    }

    // Bei gedrücktem STOP-Taster Schleife ZAEHLEN verlassen
    if (lese_bit(GPIO_Basis + GPIO_Eingabe, STOP) != 0) {
        break;
    }
}

// Uhr stoppen (Timer-Interrupt sperren)
loesche_bit(Timer_Basis + Timer_Kontroll, Timer_IrEn);

} // Ende der Schleife ZAEHLEN
} // Ende des Hauptprogramms
```

Aufgabe 1: Erstellung und Test der Peripheriekomponente "SSP"

Lernziele

- Erstellen einer Beispielrechner-Komponente

Aufgabenstellung

Erstellen und testen Sie die Peripheriekomponente *SSP*. Verwenden Sie dabei die aus einem vorherigen Versuch bereits bekannte Komponente *Siebensegment_Anzeige*.

Durchführung

1. Entpacken Sie die bereitgestellte Verzeichnisstruktur in einem Arbeitsverzeichnis. Achten Sie darauf, dass dessen Pfad keine Leer- oder Sonderzeichen wie zum Beispiel Umlaute enthält.
2. Vervollständigen Sie die Datei `Hardware\Peripherie\SSP.vhd`. Ergänzen Sie im Deklarationsteil der Architecture die benötigten Signale. Beschreiben Sie im Anweisungsteil den Adressdecoder, die vier Register und den Lesedatenmultiplexer. Erzeugen Sie eine Instanz der Komponente *Siebensegment_Anzeige* (Übernehmen Sie dabei den Wert des generischen Parameters `MUX_CYCLES` aus der entity).
3. Starten Sie den VHDL-Simulator (ModelSim oder QuestaSim) und ändern Sie dessen Arbeitsverzeichnis zu *Hardware/Peripherie*. Verifizieren Sie die von Ihnen erstellte Komponente mit der bereitgestellten Testbench `SSP_testbench.vhd` und der bereitgestellten Skriptdatei `test_SSP.do`. Beachten Sie dabei die Ausgaben im Transcript-Fenster. Zur Beschleunigung des Tests wird das Multiplexing der Siebensegment-Anzeigen in der Testbench durch einen kleinen Wert für den generischen Parameter `MUX_CYCLES` beschleunigt.

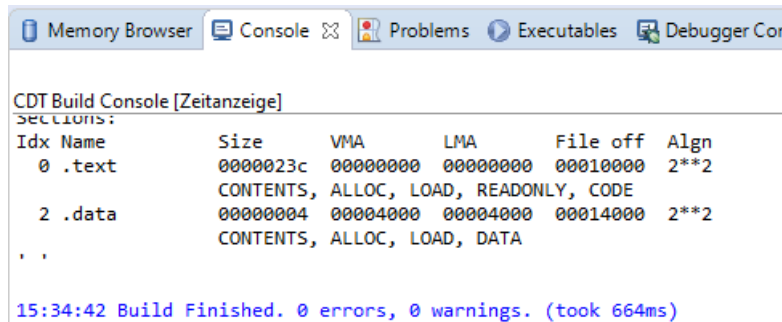
Aufgabe 2: Software vervollständigen

Lernziele

- Programmieren von Peripheriezugriffen

Durchführung

1. Starten Sie Eclipse. Verwenden Sie als Workspace das Verzeichnis *Software* im Arbeitsverzeichnis.
2. Importieren Sie das Projekt „Stoppuhr“ in den Workspace. Dieses enthält im Verzeichnis *source* die Assembler-Datei `Stoppuhr.S`.
3. Kopieren Sie die aus Versuch 3 bekannten Unterprogramme in das Programm (Stellen mit TODO markiert).
4. Ergänzen Sie im Programm das unvollständige Unterprogramm `Zeit_anzeigen` (Funktion: siehe C-Code, Stelle mit TODO markiert). Dieses Unterprogramm soll die per Register übergebenen Werte für Minuten (`$a0`), Zehner- (`$a1`) und Einer-Sekunden (`$a2`) sowie die Zehntel-Sekunden (`$a3`) in die Hardwareregister der Komponente SSP schreiben (`SSP_Wert3`, `SSP_Wert2`, `SSP_Wert1` und `SSP_Wert0`). Bei den Minuten und den Einer-Sekunden soll dabei der Dezimalpunkt gesetzt werden (siehe C-Code). Beheben Sie alle Syntaxfehler. Sie finden die Fehler- und Erfolgsmeldungen in der Console:



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output is from the 'CDT Build Console [Zeitanzeige]' and displays a table of sections for the build. The table has columns for Index, Name, Size, VMA, LMA, File offset, and Alignment. Two sections are listed: '.text' at index 0 and '.data' at index 2. Below the table, a status message indicates the build finished successfully with 0 errors and 0 warnings, taking 664ms.

```
CDT Build Console [Zeitanzeige]
SECTIONS:
Idx Name      Size      VMA      LMA      File off  Algn
 0 .text      0000023c 00000000 00000000 00010000 2**2
          CONTENTS, ALLOC, LOAD, READONLY, CODE
 2 .data      00000004 00004000 00004000 00014000 2**2
          CONTENTS, ALLOC, LOAD, DATA
, ,

15:34:42 Build Finished. 0 errors, 0 warnings. (took 664ms)
```

5. Die Timer-Komponente versucht, in regelmäßigen Abständen den Prozessor zu unterbrechen. Diese Unterbrechung soll im realen System einmal pro ms erfolgen. Die globale Variable `ms` müsste dann bei jedem Timer-Interrupt um den Wert 1 erhöht werden. Der Interrupt-Handler müsste 100-mal aufgerufen werden, bis sich der Zehntel-Wert einmal ändert. Zur Beschleunigung der anschließenden VHDL-Simulation wird zum einen der Reload-Wert des Timers so konfiguriert, dass nicht im Abstand von 1 ms, sondern im Abstand von 50 μ s ein Interrupt ausgelöst wird (Beschleunigungsfaktor 20). Außerdem wird im Timer-Handler die Variable `ms` nicht um den Wert 1, sondern um den Wert 50 erhöht (Beschleunigungsfaktor 50). Beide Maßnahmen zusammen bewirken für das simulierte System eine Beschleunigung der Zeitanzeige um den Faktor 1000 gegenüber dem realen System. Zur Konfiguration dieser Werte dienen die Konstanten im oberen Teil des Programms:

```
# Konstanten fuer die Ausfuehrung in Hardware (Echtzeit)
.set ZAEHLER_PERIODE, 59999 # Echtzeit (Ein Interrupt pro ms)
.set MS_PRO_INTERRUPT, 1    # Echtzeit (1 ms pro Interrupt)

# Konstanten fuer die Ausfuehrung im VHDL-Simulator (Beschleunigung x1000)
.set ZAEHLER_PERIODE, 2999 # Beschleunigt (20 Interrupts pro ms)
.set MS_PRO_INTERRUPT, 50  # Beschleunigt (50 ms pro Interrupt)
```

Stellen Sie sicher, dass die Echtzeit-Parameter aus- und die Simulations-Parameter nicht auskommen-tiert sind

6. Sehen Sie sich die Datei [Stoppuhr_diss.txt](#) (Im Ordner Debug des Eclipse-Projekts). Diese enthält unter anderem die Symboltabelle des Programms:

```
10 SYMBOL TABLE:
11 00000000 l d .text 00000000 .text
12 00004000 l d .data 00000000 .data
13 00004000 l .data 00000000 ms
```

Weiter unten in dieser Datei finden Sie auch den Maschinencode des Programms.

7. Sehen Sie sich die Datei [Stoppuhr.hex](#) an (ebenfalls im Ordner Debug). Diese enthält das Programm im Intel-Hex-Format, es wird später zur Initialisierung der Speicher verwendet.
8. Exportieren Sie abschließend noch das Programm für die VHDL-Simulation, indem Sie in Eclipse im „Project Explorer“ auf die Batch-Datei [CopyHexFile.bat](#) durch Doppelklick ausführen. Dadurch wird die Datei [Stoppuhr.hex](#) in das Verzeichnis *Hardware/Speicher* umkopiert und dort zu [Software.hex](#) umbenannt. Bei der anschließenden VHDL-Simulation wird diese Datei eingelesen und mit deren Inhalt die Speicher initialisiert.

Aufgabe 3: System im VHDL-Simulator simulieren

Lernziele

- Einbinden der erstellten Komponente in das Beispielrechner-System
- Simulation des Systems im VHDL-Simulator

Durchführung

1. Passen Sie die Datei [Hardware\Beispielrechner_System.vhd](#) an die Systemarchitektur an. Erstellen Sie dazu eine Instanz der Komponente *SSP*. Für deren generischen Parameter `MUX_CYCLES` verwenden Sie den generischen Parameter `SSP_MUX_CYCLES` aus der Entity. Ergänzen Sie die neuen Ausgangssignale (*SEG und MUX*) in der Portbeschreibung. Ergänzen Sie die benötigten Signale und Verbindungen für die Busanbindung. Den Adressraum der Komponente *SSP* können Sie der Software [Stoppuhr.S](#) entnehmen. Die von Ihnen zu bearbeitenden Stellen sind mit TODO gekennzeichnet.
2. Starten Sie den VHDL-Simulator (ModelSim oder QuestaSim). Stellen Sie dessen Arbeitsverzeichnis auf das Verzeichnis „Hardware“ im Arbeitsverzeichnis ein. Übersetzen und simulieren Sie das System mittels der vorgegebenen Kommandodatei [test_Beispielrechner_System_V4.do](#). Beachten Sie dabei die Ausgaben im Transcript-Fenster. Zur Beschleunigung des Tests wird auch hier das Multiplexing der Siebensegment-Anzeigen in der Testbench durch einen kleinen Wert für den generischen Parameter `MUX_CYCLES` beschleunigt.
3. Suchen Sie im Wave-Fenster einen Aufruf des Unterprogramms *Zeit_anzeigen*. Suchen Sie darin einen schreibenden Zugriff auf das Register *Wert1* der Komponente *SSP*. Machen Sie von diesem Ausschnitt des Wave-Fensters einen PDF-Ausdruck ([Zeit_anzeigen.pdf](#)).
4. Suchen Sie im Wave-Fenster einen Aufruf des Unterprogramms *Timer_Handler*. Vermessen Sie per Cursor die Dauer einer Interrupt-Bearbeitung. Machen Sie von diesem Ausschnitt des Wave-Fensters einen PDF-Ausdruck ([Interrupt.pdf](#)).

Aufgabe 4: Synthese und Test mit realer Hardware

Lernziele

- Synthetisieren des Beispielrechner-Systems
- Ausführen von Programmen in realer Hardware

Durchführung

1. Starten Sie Vivado und öffnen Sie das Projekt [Synthese.xpr](#) im Verzeichnis *Synthese*. Ergänzen Sie in der Constraints-Datei [Beispielrechner_System.xdc](#) die Einträge für die neu hinzugekommenen Signale:


```
set_property -dict {PACKAGE_PIN U7 IOSTANDARD LVCMOS33} [get_ports {SEG[0]}]
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports {SEG[1]}]
set_property -dict {PACKAGE_PIN U5 IOSTANDARD LVCMOS33} [get_ports {SEG[2]}]
set_property -dict {PACKAGE_PIN V8 IOSTANDARD LVCMOS33} [get_ports {SEG[3]}]
set_property -dict {PACKAGE_PIN U8 IOSTANDARD LVCMOS33} [get_ports {SEG[4]}]
set_property -dict {PACKAGE_PIN W6 IOSTANDARD LVCMOS33} [get_ports {SEG[5]}]
set_property -dict {PACKAGE_PIN W7 IOSTANDARD LVCMOS33} [get_ports {SEG[6]}]
set_property -dict {PACKAGE_PIN V7 IOSTANDARD LVCMOS33} [get_ports {SEG[7]}]

set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports {MUX[0]}]
set_property -dict {PACKAGE_PIN U4 IOSTANDARD LVCMOS33} [get_ports {MUX[1]}]
set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports {MUX[2]}]
set_property -dict {PACKAGE_PIN W4 IOSTANDARD LVCMOS33} [get_ports {MUX[3]}]
```
2. Lassen Sie von Vivado eine Bitstream-Datei erzeugen. Schalten Sie das BASYS3-Board ein und übertragen Sie mit dem Vivado-Hardware-Manager die Bitstream-Datei auf das FPGA.
3. Starten Sie den BSR2-GDB-Server.
4. Ändern Sie in Eclipse im Programm [Stoppuhr.S](#) die Konstanten für die beschleunigte Ausführung, so dass die langsameren Werte für die Ausführung in der Hardware verwendet werden.
5. Erzeugen Sie in Eclipse eine neue Debug Configuration für das Projekt Stoppuhr. Die Einstellungen entsprechen denen in Versuch 3.
6. Führen Sie das Programm mit dem Debugger aus und überprüfen Sie seine Funktion:



Bitte legen Sie nach Abschluss des Versuchs folgende Dateien im OSCA-Dateibereich Ihrer Arbeitsgruppe in einem neuen Ordner „V4“ ab:

- SSP.vhd
- Beispielrechner_System.vhd
- Beispielrechner_System.xdc
- Stoppuhr.S
- Zeit_anzeigen.pdf
- Interrupt.pdf