Praktikum Softprocessor SP7: Anwendungsspezifische Peripherie

2. Juli 2014

Zusammenfassung

In dieser Übung werden zwei in VHDL entwickelte periphere Schaltungen an den bisherigen eingebetteten Rechner angeschlossen, eine Multiplex-Ansteuerung für die 4-stellige Siebensegmentanzeige auf der Baugruppe und eine spezielle Schaltung zur Bestimmung der Temperatur aus der PWM-Ausgabe des Sensors SMT160. Für den so konfigurierten Rechner sind zum einen Programmieraufgaben zu lösen und zum anderen sollen auch zwei eigene anwendungsspezifische periphere Einheiten entwickelt, an den Prozessor angeschlossen und in zu entwickelnden Programmen angesprochen werden. Das ist zum eine Schaltung zur Drehzahlmesung für den Lüfter und zum anderen eine True-Color-Ansteuerung für die RGB-Leuchtdiode über PWM.

1 Anwendungsspezifische Peripherie

In den vergangen Praktika wurden die peripheren Standardeinheiten eines Rechners

- parallele Schnittstelle
- serielle Schnittstelle (UART)
- Zähler/Zeitgebereinheit und
- Interrupt-Controller

eingeführt und verwendet. Es gibt weitere Standardeinheiten z.B. serielle Schnittstellen für andere Protokolle (SPI, I2C oder CAN). Darüber hinaus können für unseren programmierbaren Logikschaltkreis auch eigene Schaltungen mit beliebiger Funktion entworfen und an den Softprozessor angeschlossen werden. Für die Ankopplung an den Softprozessor soll nur der einfachste Fall betrachtet werden, die Ankopplung als PLB-Slave. Bei einem PLB-Slave sieht der Prozessor die periphere Einheit als eine Menge adressierbarer Register, die vom Programm aus gelesen und/oder beschrieben werden können. Eine Erweiterung um einen Interruptausgang oder für größere zu transferierende Datenmengen die Kopplung die Ankopplung über FIFO- oder Blockspeicher sind möglich.

Eine anwendungsspezifische periphere als PLB-Slave angeschlossene Einheit ist eine Schaltung mit

- Schaltkreisanschlüssen als Ein- und Ausgänge,
- vom Prozessor beschreibbaren Registern als Eingänge und
- Ausgängen, deren Werte in Registern gehalten und die der Prozessorgelesen werden.

Der Entwurf der anwendungsspezifischen Peripherie erfolgt ganz normal mit ISE. Zur Simulation ist die Entwurfseinheit wie üblich in einen Testrahmen einzubetten. Der Testrahmen muss die

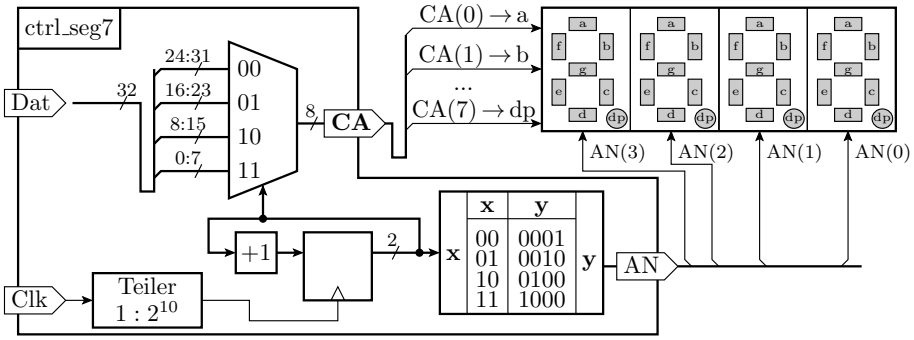


Abbildung 1: Test der 7-Segmentanzeige

Schreiboperationen der prozessorseitigen Schnittstellenregister und die externen Eingaben nachbildet und optional die Rückgaben an den Prozessor, sowie die internen Zustände und Ausgaben kontrollieren oder protokollieren. Für unsere beiden kleinen Entwurfsbeispiele bietet sich weiterhin an, die Schaltungsentwürfe, bevor sie an den Softprozessor angeschlossen werden, über die Schalter, Leuchtdioden und den Logikanalysator auf der Versuchsbaugruppe zu testen.

1.1 7-Segmentausgabe

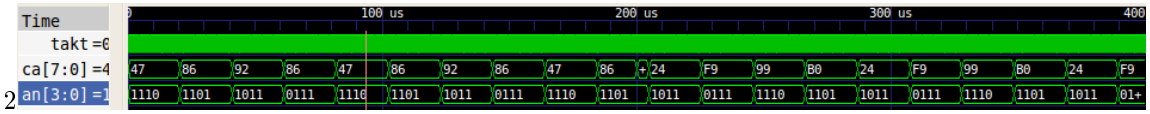
Die 7-Segmentanzeige auf der Versuchsbaugruppe hat vier Ziffern. Die Kathodensignale aller Segment a bis g und dp sind zur Einsparung von Leitungen miteinander verbunden. Die vier Anodensignale »AN0« bis »AN3« jeder Ziffer sind einzeln herausgeführt und werden auf der Baugruppe außerhalb des FPGAs mit einem Transistor invertiert (Abb. 1). Damit ein Anzeigeelement leuchtet, muss das zugehörige Kathodensignal, d.h a, b, ... oder dp auf null und das zugehörige Anodensignal, d.h. »AN0«, »AN1«, ... oder »AN3«, auch auf null gesetzt werden. Um vier Ziffern darstellen zu können, müssen diese zyklisch immer nacheinander angezeigt werden. Die zu entwerfende periphere Einheit soll ein 32-Bit-Eingabesignal haben, das von einem Ausgaberegister des Prozessors geliefert wird. In jedem Byte i soll der Rechner das Ausgabemuster für die 7 Segmente und den Punkt der Stelle i schreiben. Die Multiplexausgabe soll wie in der vierten Aufgabe des Praktikums digitaler Schaltungsentwurf mit dem durch 2^{10} geteilten Baugruppentakt von etwa 50kHz erfolgen (Abbildung 1).

Die Schnittstelle der zu entwerfenden Einheit ist:

```
entity ctrl_seg7_x is
 port( Dat: in std_logic_vector(31 downto 0);
   Clk: in std_logic;
   AN: out std_logic_vector(3 downto 0);
   CA: out std_logic_vector(7 downto 0));
end entity;
```

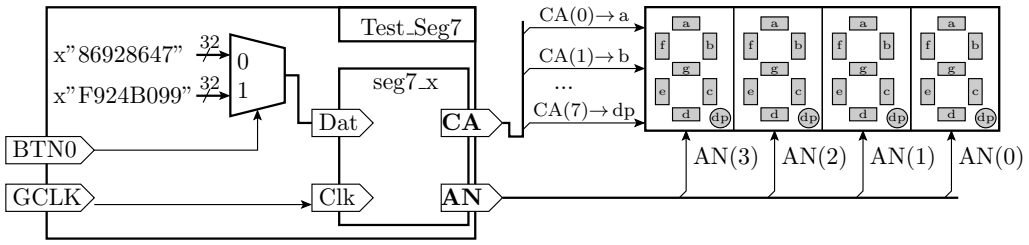
Nach dem Entwurf ist die Schaltung zu simulieren. Der Testrahmen zur Simulation muss die Eingabedaten und den Takt bereitstellen:

```
entity Testrahmen_Seg7 is  end entity;
architecture sim of Testrahmen_Seg7 is
 signal Dat: std_logic_vector(31 downto 0);
 signal Takt: std_logic = '0';
 signal AN: std_logic_vector(3 downto 0);
 signal y:  std_logic_vector(7 downto 0);
begin
 <Instanziierung des Testobjekts>
```



2

Abbildung 2: Test der 7-Segmentanzeige



Ausgangssignalname	GCLK	BTN0	a	b	c	d	e	f	g	dp	AN(0)	AN(1)	AN(2)	AN(3)
Schaltkreisanschluss	T9	M13	E14	G13	N15	P15	R16	F13	N16	P16	D14	G14	F14	E13

Abbildung 3: Test der 7-Segmentanzeige

```
Taktprozess: process
begin
  while now < 400 us loop
   wait for 10 ns; Takt <= not Takt;
  end loop;
end process;
Dat <= x"86928647", x"F924B099" after 200 us;
end architecture;
```

Abbildung 2 zeigt das Simulationsergebnis mit ghdl. Der Takt wechselt immer nach einer Millisekunden. Das Signal »Dat« wechselt genau mit der sechsten steigenden Taktflanke. Es ist insbesondere zu kontrollieren, das immer der Bytewert des Datenwortes ausgegeben wird, dessen zugeordnetes Bit im Signalvektor »AN« null ist.

Um sicher zu gehen, dass die entworfene Schaltung tatsächlich richtig funktioniert, kann sie auch auf der Baugruppe in eine einfache Testschaltung einbettet und ausprobiert werden. In der Testschaltung in Abbildung 3 kann mit BTN0 zwischen den beiden simulierten Datenwertkonstanten umgeschalten werden. Wenn die zu testende Schaltung, die Rahmenschaltung und die ucf-Datei korrekt sind, sollte bei ungedrücktem Tester das Wort »ESEL.« mit abschließenden Punkt und bei gedrücktem Taster die Zeichenfolge »12.34« angezeigt werden. Es gibt auch beim Entwurf einer so kleinen Schaltung genügend Fehlermöglichkeiten, so dass das sicher nicht beim ersten Versuch klappen wird.

1.2 Temperaturmessschnittstelle für den Sensor SMT160

Auf der Ansteckbaugruppe befindet sich zusätzlich hinter dem Lüfter ein Temperatursensor vom Typ SMT160. Die gemessene Temperatur wird als PWM-Signal kodiert ausgegeben. Laut Datenblatt besteht zwischen der Temperatur und der relativen Pulsbreite des PWM-Signals der Zusammenhang:

$$DC_0 = 0,32 + 0,0047 \cdot T$$

(T – Temperatur in °C). Die zu berechnende Temperatur beträgt in Abhängigkeit von der Pulsbreite:

$$T = \frac{DC_0 - 0,32}{0,0047} \tag{1}$$

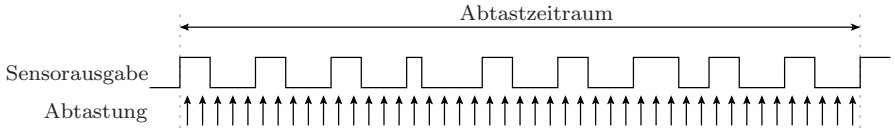


Abbildung 4: Abtastung Temperatursensor

Zur Berechnung des Tastverhältnisses DC_0 muss zunächst das Ausgabesignal des Temperatursensors mit einer ausreichend hohen Frequenz und über einem genügend langen Zeitraum abgetastet werden (Abbildung 4).

Dabei werden die Anzahl der Low-Pegel gezählt und anschließend durch die Anzahl aller Abtastpunkte geteilt. In Abbildung werden 45 Werte abgetastet und 27 mal ist der Abtastwert Null. Eingesetzt in obige Formel ergibt sich daraus das Tastverhältnis

$$DC_0 = \frac{n_0}{N} = \frac{27}{45} = 0,6$$

und die Temperatur:

$$T = \frac{0,6 - 0,32}{0,0047} \approx 60°C$$

Die zu entwerfende Schnittstelle soll aus einem les- und beschreibbaren Register Reg0 und einem nur lesbarenbaren Register Reg1 bestehen. Das erste Register soll, in jedem Takt, wenn es nicht beschrieben wird oder null ist, abwärts zählen. Das Register »Reg1« soll beim beschreiben des ersten Registers auf null gesetzt und sonst, wenn das andere Register noch nicht wieder null ist und das Eingangssignal vom Sensor '1' ist, aufwärtszählen (Abbildung 5 a). Abbildung 5 b zeigt das Ergebnis einer Beispielsimulation. Als Taktfrequenz wurde 10 kHz gewählt und das Sensorsignal hat eine relative Pulsbreite von 0,3. Die Periode des Sensorsignals beträgt im Mittel 250 µs und ist mit einer Zufallszahl überlagert, um mögliche Störungen mit nachzubilden. Nach der Beschreibung der zu entwerfenden Schaltung in VHDL ist ein Testrahmen zu programmieren, der die Signale wie in der Abbildung bereitstellt und zu kontrollieren, das der eigenen Entwurf genauso funktiuoniert.

Auch wenn die Simulation erfolgreich durchläuft, kann die entworfende Schaltung immer noch fehlerhaft sein. Für das Beispiel ist ein Test mit dem Logikanalysator zu empfehlen. Dazu ist die Schaltung in eine Rahmenschaltung einzubetten, die die Eingabesignale, die nicht vom Sensor kommen (Clk, WR und DW0) bereitstellt und die zu kontrollierenden Signale auf Schaltungsausgänge führt, die sich an den Logikanalysator anschließen lasst.

Für den Testrahmen in Abbildung 6 ist die Lüfterbaugruppe an den Erweiterungsstecker A1 und wie im Praktikum Digitaler Schaltungsentwurf 1, Aufgabe 2 ein »Digilent Test Point Header« an den Erweiterungsstecker A2 und an diesen der USB-LOGI-500 anzuschließen. Um die Ausgabe des Logikanalysators auswerten zu können, soll mit einem reduzierten Takt von etwa 10 kHz und einem Schreibpuls aller 10 ms und einer Konstanten für DW0 getestet werden. Die zu beobachtenden Signale sollen wie in der Abbildung gezeigt, an die Anschlüsse des Logikanalysator LA(0) bis LA(15) angeschlossen werden. Abbildung 7 zeigt die Zuordnung der Gehäuseanschlüsse für die UCF-Datei und wo die Kabel des USB-LOGI an den »Digilent Test Point Header« anzuschließen sind.

Der USB-LOGI ist mit folgender Konfigurationsdatei aufzurufen.

```
<la>
<samplerate>100000</samplerate>
<pretrigger>1</pretrigger>
<signals>
```

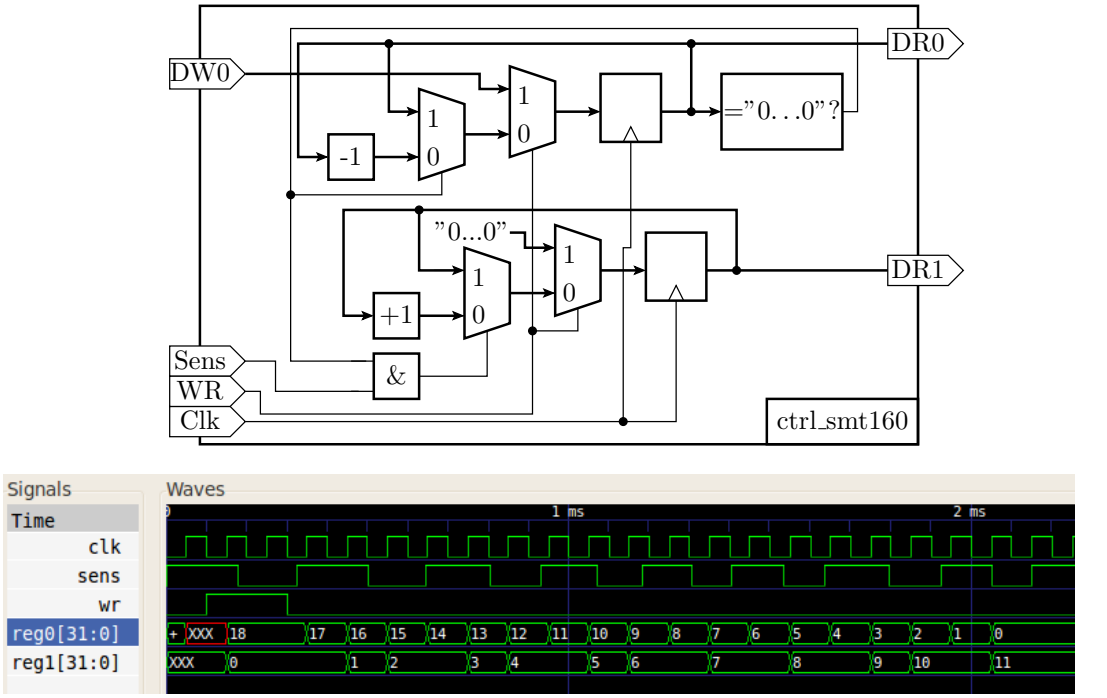


Abbildung 5: Sensorschnittstelle a) Schaltung b) Simulationsbeispiel

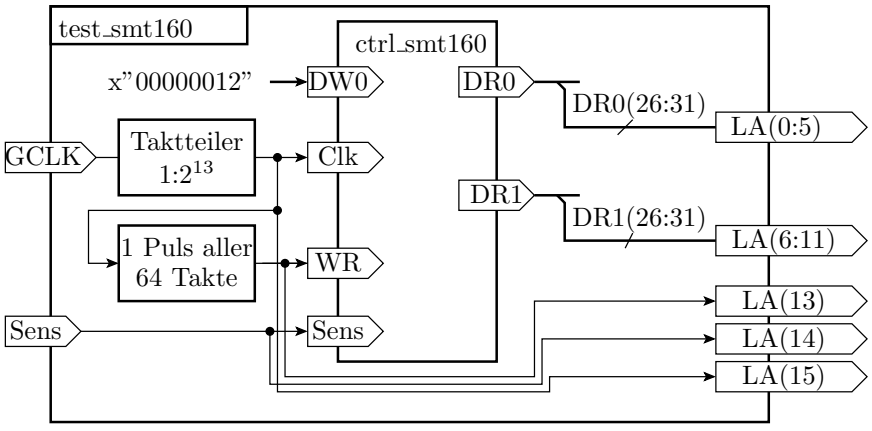
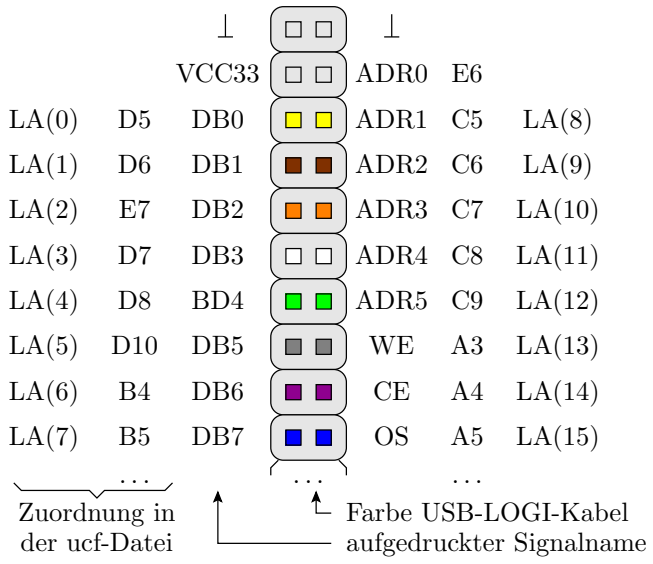


Abbildung 6: Testrahmen zum Ausprobieren der Schaltung in Abbildung 5




```
<la>
  <samplerate>100000</samplerate>
  <pretrigger>1</pretrigger>
  <signals>
    <signal name="Clk"> <ch>15</ch></signal>
    <signal name="Sens"><ch>14</ch></signal>
    <signal name="WR"> <ch>13</ch></signal>
    <signal name="DR0">
      <ch>0</ch><ch>1</ch> ... <ch>5</ch>
    </signal>
    <signal name="DR1">
      <ch>6</ch><ch>7</ch> ... <ch>11</ch>
    </signal>
  </signals>
  <trigger when="A" counter ="1">
  <A>
    <ch when="high">13</ch>
  </A>
  </trigger>
</la>
```

Abbildung 7: Sensorschnittstelle a) Schaltung b) Simulationsbeispiel

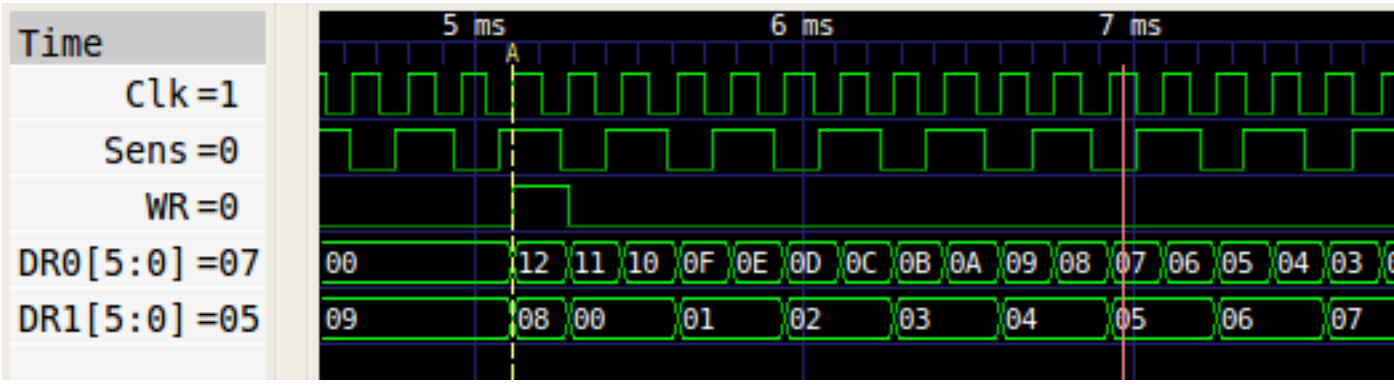
```
<signal name="Clk"> <ch>15</ch></signal>
<signal name="Sens"><ch>14</ch></signal>
<signal name="WR">  <ch>13</ch></signal>
<signal name="DR0">
<ch>0</ch><ch>1</ch> ... <ch>5</ch>
</signal>
<signal name="DR1">
<ch>6</ch><ch>7</ch> ... <ch>11</ch>
</signal>
</signals>
<trigger when="A" counter ="1">
<A>
<ch when="high">13</ch>
</A>
</trigger>
</la>
```

Wenn die Schaltung und der Testrahmen korrekt entworfen sind, sollte das vom Logikanalysator aufgezeichnete Ergebnis etwa wie in Abbildung 8 aussehen.

2 Hardware-Entwurf

Abbildung 9 zeigt das zu entwerfende Rechnersystem. Es umfasst das komplette System aus dem fünften Praktikum (Prozessor, Speicher, parallelen Schnittstellen, UART, Debugschnittstelle und Zähler/Zeigebereinheit). Zusätzlich wird an den Peripheriebus ein Interrupt-Controller angeschlossen. Über diesen werden die Interruptsignale der Zähler/Zeitgebereinheit, der UART und der Eingabeschnittstelle an den Prozessor weitergeleitet.

Kopieren Sie sich aus den bisherigen Projekten folgende Entwurfsdateien



rausnehmen

Abbildung 8: Mit dem Logikanalysator bestimmte Signalverläufe

```
.../Aufg6/data/system.ucf
.../Aufg6/system.xmp
.../Aufg6/system.mhs
.../Aufg6/system.mss
```

in ein neu anzulegendes Unterverzeichnis:

```
.../Aufg7
```

Starten Sie das Entwurfsprogramm:

```
Anwendungen ▷ Umgebung ▷ Xilinx Platform Studio (EDK)
```

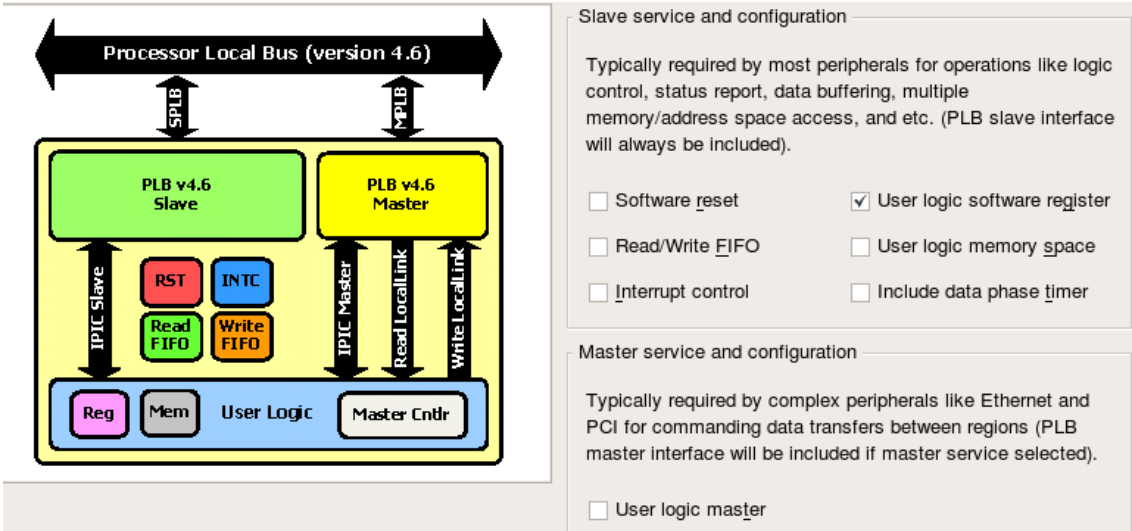
Öffnen Sie das Projekt

```
File ▷ Open Projekt ▷ <Auswahl von .../Aufg7/system.xmp>
```

2.1 Erzeugen eines Templates für die neue Peripherie

Die nachfolgende Klick-Schrittfolge ist für beide anwendungsspezifischen Peripherie-Schaltungen durchzuführen.

1. Hardware -> Create or Import Peripheral
2. Next
3. Create templates for a new peripheral, Next
4. Next
5. Namen vergeben: einmal »seg7« und beim zweite mal »sens_smt160« (nur Kleinbuchstaben verwenden)
6. Next
7. Für den anzuschließenden Bus »Processor Local Bus« (PLB vX.X) auswählen und Next
8. Benötigte Features auswählen: Hier nur »User logic software registers« auswählen. Next



9. Next
10. Als Anzahl der benötigten Register den 7-Segment-Anzeige-Controlle »1« und den Controller für den Temperatursensor »2« auswählen.
11. Einstellungen beibehalten, Next
12. Kein Simulationsmodell, Next
13. Zusätzliche Optionen auswählen: »ISE Projekt erstellen«, Next
14. Finish

Alle selbst erstellten IP-Cores liegen im Unterverzeichnis "pcores". In jedem IP-Core-Verzeichnis gibt es drei Unterverzeichnisse. Im Unterverzeichnis »data« liegen eine *.pao- (Peripheral Analysis Order) und eine *.mpd- (Microprocessor Peripheral Description) Datei. In die pao-Datei sind alle selbst erstellten VHDL-Entwurfseinheiten für den Core zu ergänzen. Für den 7-Segment-Decoder ist das nur die Datei »ctrl_Seg7.vhd«.

```
...
lib plbv46_slave_single_v1_01_a plbv46_slave_single vhdl
lib seg7_v1_00_a ctrl_seg7 vhdl  # einzufügende Zeile
lib seg7_v1_00_a user_logic vhdl
lib seg7_v1_00_a seg7 vhdl
```

Der Eintrag hat folgenden Aufbau: Schlüsselwort »lib«, dann die Biblithek, in die die analysierte Datei abzulegen ist, entity-Name der Entwurfseinheit und das Schlüsselwort »vhdl« für den Beschreibungstyp. Die Bibliothek muss dieselbe sein, in der die Entwurfseinheit »user_logic« enthalten ist und der Eintrag muss vor dem für die Entwurfseinheit »user-logic« stehen, weil sie in diese als Instanz eingebunden werden muss. Für Sensor-Controller ist in der pao-Datei vor der Zeile für »user_logic« folgende Zeile zu ergänzen:

```
lib sens_smt160_v1_00_a ctrl_smt160 vhdl
```

In der mpd- (Microprocessor Peripheral Description) Datei sind alle externen Anschlüsse und Parameter (Ports und Generics) zu ergänzen. Jede Zeile beschreibt einen Anschluss oder Anschlussbus und hat folgenden Aufbau:

```
PORT <Anschlussname> = "<Anschlusstyp>", DIR = <I|O>, [VEC = [<Indexbereich>]
```

(Anschlussname wie später in der Schnittstellenbeschreibung von »user_logic« zu eränzen; Anschlusstyp – kann Reset, Interrupt oder eine leere Zeichenkette sein). Der Indexbereich ist nur für Signalvektoren anzugeben. Für den Anzeige-Controller sind in der mpd Datei die beide Ausgangsbusse zu ergängen:

```
PORT CA = "", DIR = O, VEC = [0:7]
PORT AN = "", DIR = O, VEC = [0:3]
```

Für den Sensor-Controller ist nur das Eingangssignal für den Sensor in der mpl-Datei zu ergänzen:

```
PORT Sens = "", DIR = I
```

2.2 VHDL-Dateien für den 7-Segment-Controller anpassen

Zuerst sind die eigenen Entwurfsbeschreibungen »ctrl_Seg7.vhd« in das Verzeichnis »../hdl/vhdl/« des Cores zu kopieren. Dann ist in diesem Verzeichnis die Datei »user_logic.vhd« zu öffnen. In der Schnittstellenbeschreibung sind die beiden Ausgabebusse zwischen den dafür vorgesehenen Kommentarzeilen zu ergänzen:

```
 -- ADD USER PORTS BELOW THIS LINE ------------------
CA: out std_logic_vector(7 downto 0);
AN: out std_logic_vector(3 downto 0);
--USER ports added here
```

Zu Beginn der Funktionsbeschreibung ist die selbst entwickelte Entwurfseinheit als Instanz einzufügen:

```
  --USER logic implementation added here
Seg7_Controller: entity ctrl_seg7(A)
  port map(CA => CA, AN => AN, Dat => slv_reg0, Clk => Bus2IP_Clk);
```

Die Anschlüsse für die 7-Segmentanzeige werden mit den gleichnamifen Schaltungsausgängen verbunden, der Dateneingang mit dem Register »slv_reg0« und der Takt mit dem PLB-Bustakt. Der Schreibprozess »SLAVE_REG_WRITE_PROC« setzt das Register »slv_reg0« bei einem Bus-Reset auf null (alle Ausgabeelemente leuchten) und schreibt sonst, wenn das Schreibauswahlsignal »slv_reg_write_sel« "1" und das zugehörige Byte-Auswahlsignal aktiv ist, den Datenwert vom Bus in das Register. Er kann unverändert übernommen werden. Der Leseprozess für das Register »SLAVE:_REG_READ_PROC« zum zurücklesen des Registerinhalts wird nicht gebraucht.

- ——
- devl: Hier liegt das erzeugte ISE Projekt (im Unterordner projnav), was zum Programmieren des IP-Cores hilfreich ist
- hdl: Hier liegen die vhdl-Quellen des Cores
  - <Name des IP Cores>.vhd: Top-Level Datei des Cores, enthält Schnittstellenbeschreibung zum PLB Bus
    - In dieser Datei dürfen an den markierten Stellen (z.B. “ADD USER PORTS BELOW THIS LINE”) nur externe Ports und Generic-Parameter hinzugefügt werden
    - Die Datei instanziert eine Einheit namens “plbv46_slave_single”, welche eine Umsetzung der komplizierten PLB Bus-Signale auf intuitiver zu verwendende Signale durchführt, die von der user_logic (siehe unten) verwendet werden
  - user_logic.vhd: Enthält Beispielcode zum Lesen und Schreiben von Registern, hier muss der eigene VHDL Code mit rein

- Die Kommentare in dieser Datei sind hilfreich beim Verständnis
- Es gibt einen Prozess zum Schreiben (SLAVE_REG_WRITE_PROC) und einen Prozess zum Lesen (SLAVE_REG_READ_PROC) von Registern (Lesen/Schreiben aus Sicht des C-Programms!)
- Der Prozess zum Lesen kann i.A. unverändert übernommen werden
- Der Prozess zum Schreiben muss dann verändert werden, wenn der Core selbst Register beschreiben soll (aus Sicht des C-Programms wäre das ein "nur lesend" Register)

```
entity user_logic is   generic   (
 C_SLV_DWIDTH  : integer   := 32; -- kann man nicht ändern
 C_NUM_REG     : integer     := 1);
 port   ( - Bus protocol ports, do not add to or delete
   Bus2IP_Clk    : in  std_logic;    -- PLB-Takt 50 MHz
  Bus2IP_Reset    : in  std_logic;
   Bus2IP_Data    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
   Bus2IP_BE      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
  Bus2IP_RdCE    : in  std_logic_vector(0 to C_NUM_REG-1);
  Bus2IP_WrCE     : in  std_logic_vector(0 to C_NUM_REG-1);
  IP2Bus_Data    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
  IP2Bus_RdAck    : out std_logic;  -- hält den Bus an
   IP2Bus_WrAck    : out std_logic;
   IP2Bus_Error   : out std_logic);
```

2.3 IP Core Templates anpassen (am Beispiel des Temperatursensors)

IP Core besteht aus:

- Einem Kontroll- und einem Daten-Register
- Taktteiler
- Abtastung des Sensorausgangs

Das Kontrollregister enthält ein Bit zur Aktivierung der Sensorabtastung und einen 24 Bit Wert, der die Gesamtzahl der Abtastpunkte angibt. Das Daten-Register enthält die Anzahl der gezählten low-Pegel innerhalb des Abtastzeitraums.

Zuerst VHD-Datei in Modelsim-Verzeichnis
#kennt ghdl numeric_std? testen
Änderungen in den VHDL-Dateien hier beschreiben
eigene Datein in den jeweiligen HDL-Ordner kopieren
Projekt > rescan > user repositries
Connection »Make External«
UCF anpasen
Generate Adresses

2.4 Schreiben eines Simulationsmodells

Es empfielt sich, die eigentliche Implementierung des Cores in einer oder mehreren eigenen VHDL Dateien abzulegen, diese dann in der user_logic einzubinden und mit den Registern zu koppeln. Das hat unter anderem den Vorteil, dass der Core separat simuliert werden kann. Im Beispiel ist die Implementierung in der Datei sensor_main.vhd, ein Simulationsmodell existiert nicht (da das Beispiel sehr einfach ist...).

```
entity sensor_main is
    port(
        clk             : in    std_logic;                      -- PLB Bus Clock
        rst             : in    std_logic;                      -- PLB Bus Reset (synchron!)
        sensor_output   : in    std_logic;                      -- Ausgabesignal des Sensors
        en              : in    std_logic;                      -- IP Core eingeschaltet
        samples         : in    std_logic_vector(0 to 23);      -- Anzahl der Abtastpunkte
        low             : out   std_logic_vector(0 to 23);      -- Anteilige Anzahl an low Pege
        intr            : out   std_logic                       -- Interrupt nach Beendigung de
    );
end entity;
```

2.5 Einbinden der Implementierung in die user_logic

Wenn die Implementierung ausreichend simuliert worden ist, werden alle benötigten Dateien in den Ordner hdl/vhdl Ordner kopiert (hier also nur die Datei sensor_main.vhd). Dann kann die Datei in ISE dem Projekt mit "Add Source..." hinzugefügt werden. Dabei ist darauf zu achten, die Datei(en) NICHT in die Library "work" aufzunehmen, sondern in die Library, die den selben Namen wie der IP Core hat.

Jetzt die user_logic.vhd öffnen und hinter der Kommentarzeile "–USER logic implementation added here", die eigene Entity Instanzieren.

```
--USER logic implementation added here
  sensor_main_impl : entity sensor_main
  port map(
      clk             => Bus2IP_Clk,
      rst             => Bus2IP_Reset,
      sensor_output   => sensor_output,
      en              => slv_reg0(0),
      samples         => slv_reg0(8 to 31),
      low             => slv_reg1(8 to 31),
      intr            => intr
  );
```

Die Signale sensor_output und intr werden wiederum als Port in der user_logic hinzugefügt (bei der Zeile "–USER ports added here.").

Für die Signale en, samples und low werden die automatisch erzeugten Registersignale slv_reg0 und slv_reg1 verwendet. Da slv_reg1 ein "readonly" Register ist und nur vom Core selbst beschrieben werden darf, muss der entsprechende VHDL Code zum Beschreiben dieses Registers aus dem Prozess "SLAVE_REG_WRITE_PROC" in der user_logic entfernt werden. Die zu entfernenden Abschnitte lauten:

```
slv_reg1 <= (others => '0');
```

und

```
when "01" =>
 for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
   if ( Bus2IP_BE(byte_index) = '1' ) then
     slv_reg1(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_in
   end if;
 end loop;
```

2.6 Anpassen der Top-Level Datei

In der Top-Level Datei müssen nun noch die neu hinzugefügten Ports aus der User-Logic an den markierten Stellen ("–USER ports mapped here" und "–USER ports added here") hinzugefügt werden. Hat man alles richtig gemacht, müsste sich die Datei nun problemlos synthetisieren lassen (so kann man dann auch ermitteln, was der eigene Core für FPGA Ressourcen verbraucht).

2.7 Anpassen der mpd- und pao-Datei

Die pao-Datei im data Ordner öffnen und wie oben beschrieben ändern. Die vor user_logic einzufügende Zeile lautet:

```
lib temp_sensor_v1_00_a sensor_main vhdl
```

Ebenso wie beschrieben die mpd-Datei anpassen. Die nach "## Ports" einzufügenden Zeilen lauten:

```
PORT sensor_output = "", DIR = I
PORT intr = "", DIR = O, SIGIS = INTERRUPT, SENSITIVITY = EDGE_RISING
```

2.8 Aktualisieren des EDK-Projektes

Im Menü auf "Project > Rescan User Repositories", damit die neue mpd-Datei eingelesen wird. Ansonsten alles genau wie bei anderen IP Cores auch (mit PLB verbinden, Ports verdrahten, Adressen zuweisen).

3 IP Core ändern und aktualisieren

Werden die vhdl Quellen des IP Core geändert, müssen die pao- und mpd-Datei ggf. auch angepasst werden. Bei "Rescan User Repositories" können dann Fehler in der system.mhs auftreten, die dann manuell korrigiert werden müssen (z.B. beim entfernen eines Ports).

Damit der Core dann auch neu synthetisiert wird (ohne "Clean Hardware"), müssen die entsprechenden *.ngc Synthesedateien des Cores manuell gelöscht werden. Diese befinden sich im Unterordner implementation bzw. implementation/cache des EDK-Projektes (es reicht wirklich nur die zwei *.ngc Dateien zu löschen, der Rest kann bleiben).

4 Treiber für den Core schreiben

Zum Zugriff auf die Register werden die Funktionen aus der Datei xio.h benötigt. Im Speziellen sind dies:

```
XIo_Out32(OutputPtr, Value)
```

Zum Schreiben eines 32 Wertes (Value) an eine bestimmte Adresse (OutputPtr).

```
XIo_In32(InputPtr)
```

Zum Lesen eines 32 Bit Wertes an einer bestimmten Adresse (InputPtr).

Das erste Register des IP Cores liegt immer an der Basis-Adresse (BaseAddress), welche in EDK per "Generate Addresses" vergeben wurde. Das zweite Register liegt direkt dahinter (also BaseAddress + 4) usw.

Ansonsten sind der Fantasie beim Schreiben des Treibers keine Grenzen gesetzt. Wenn möglich sollten jedoch Makros (#define-Anweisungen) gegenüber Funktionen vorgezogen werden, wann immer dies sinnvoll erscheint.

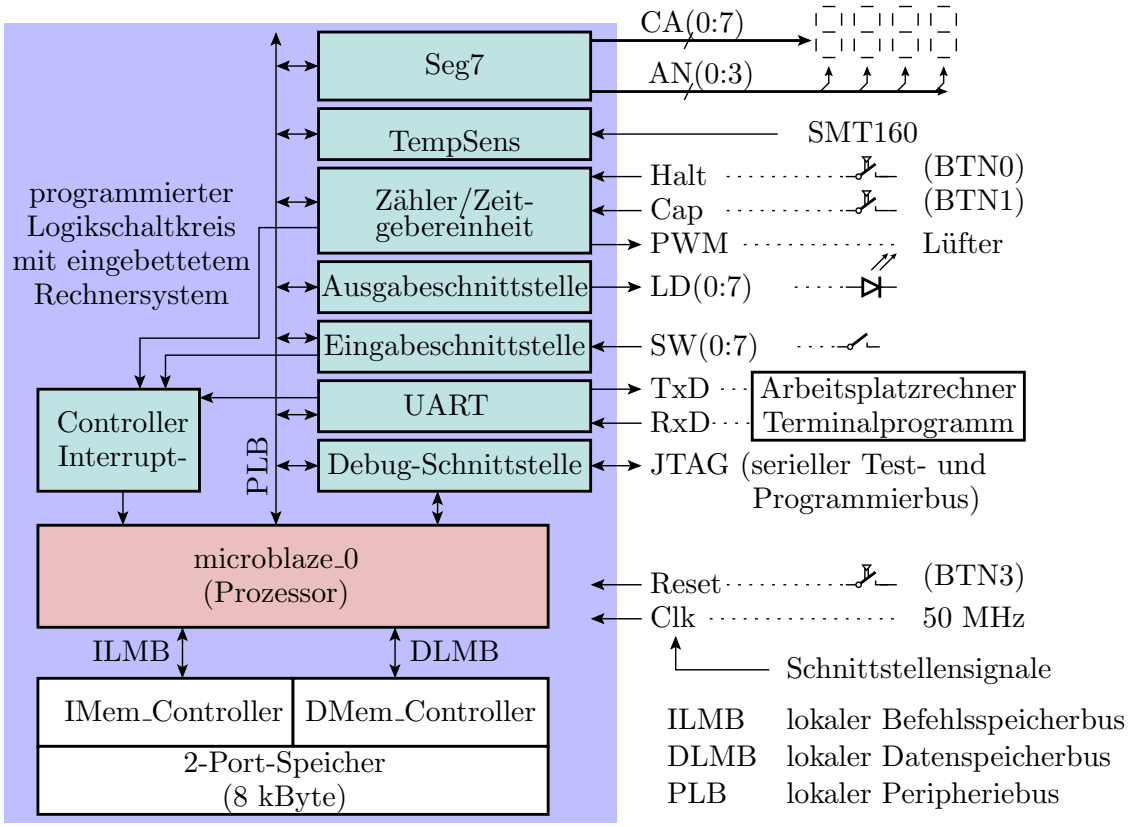


Abbildung 9: Rechnersystem mit anwendungsspezifischer PeripherieInterrupt-Controller