Aufgabe 3: Lichtsensor

G. Kemnitz, C. Giesemann, TU Clausthal, Institut für Informatik

13. Dezember 2017

Zusammenfassung

Für das Auslesen der Werte aus dem Lichtsensormodul PmodALS über dessen SPI-Schnittstelle werden Simulationsmodelle untersucht, simuliert, der SPI-Master synthetisiert, getestet, mit dem Logikanalysator untersucht und weiterentwickelt. Lernziel ist das Kennenlernen wichtiger Schritte beim Entwurf von Schaltungen für den seriellen Datenaustausch mit modernen integrierten Schaltkreisen über vorgegebene Protokolle.

1 Schaltung und Ansteuerung des PmodALS

Abb. 1 a zeigt die Schaltung des Lichtsensormoduls PmodALS. Der Fototransistor ALS1 vom Typ TEMT6000X01 erzeugt einen zur Beleuchtungsstärke proportionalen Strom, der vom Widerstand R_1 in eine Spannung am Eingang »VIN« des Analog-Digital-Wandlers »IC1« umgewandelt wird. Das Auslesen des digitalisierten 8-Bit-Wertes erfolgt über das in Abb. 1 b dargestellte SPI-Protokoll. Der zu programmierende SPI-Master zum Auslesen des ADUs aktiviert zu Beginn des Datentransfers das low-aktive[1] Signal Chip-Select »#CS« ($\overline{cs}$, Variablenname in VHDL »CS«) für 16 SCL-Takte. In den Takten 4 bis 11 legt der ADU nacheinander die 8 Datenbits in absteigender Reihenfolge auf die Datenleitung »SDA«.

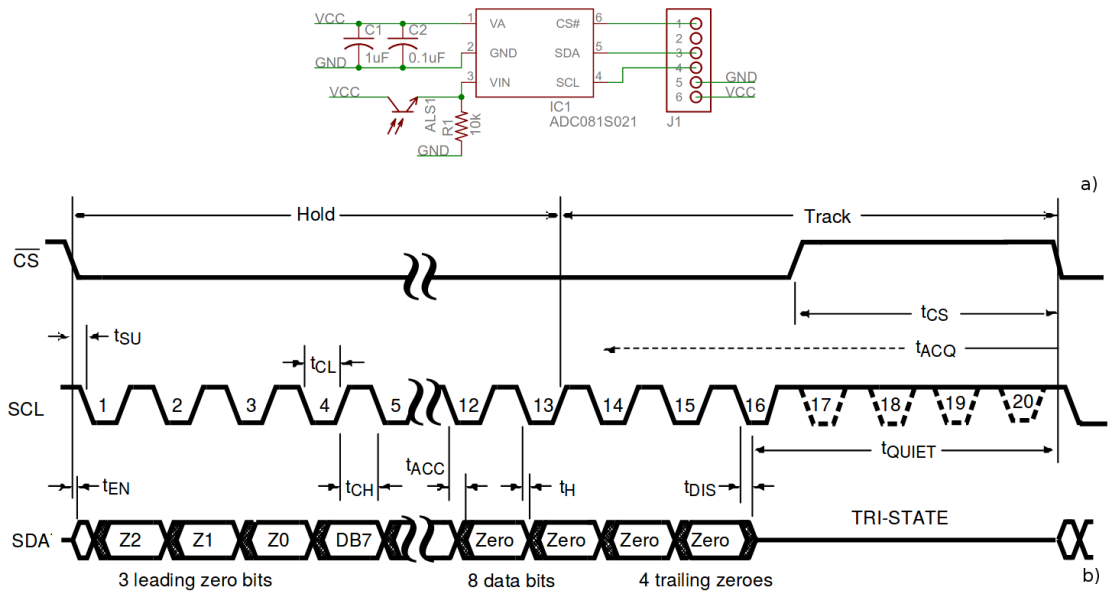

Abbildung 1: a) Schaltung des PmodALS, b) SPI-Signalverläufe

[1] Für ein low-aktives Signal bedeutet »aktivieren« das Signal auf null zu setzen.

2 Simulationsmodell für den ADC081S021

Die Simulation des zu entwerfenden SPI-Masters erfordert neben einer Testbench und einem Simulationsmodell des Testobjekts auch ein Simulationsmodell des SPI-Slaves im Analog-Digital-Umsetzer. Die digitalisierten und zu übertragenden Helligkeits-Bytewerte sollen in der Simulation durch einen Pseudo-Zufallsalgorithmus erzeugt werden. Das Simulationsmodell des SPI-Slaves muss nicht synthesefähig sein, sollte dafür aber alle Zeitbedingungen aus dem Datenblatt des ADC081S021 einhalten bzw. überprüfen. Zu prüfen sind z.B., dass der zu entwickelnde SPI-Master folgende Zeitbedingungen in Abb. 1 b einhält:

- die minimale Umwandlungsdauer t_{ACQ},
- die Mindestdauer t_{QUIET} , die »SDA« zwischen Übertragungen hochohmig ist, ...

Die Entwicklung von Verhaltensmodellen für Schaltkreise aus Datenblätter war kein Vorlesungsthema. Deshalb wird ein Beispielsimulationsmodell vorgegeben und Sie sollen untersuchen, wie gut das Modell mit der Beschreibung im Datenblatt übereinstimmt.

```
1  ------------------------------------------------------------------------
2  -- ADC081S021.vhd (SPI-Slave)
3  ------------------------------------------------------------------------
4  library IEEE; use IEEE.STD_LOGIC_1164.ALL;
5  entity ADC081S021 is
6   generic(
7    tSCL: delay_length := 500 ns;     -- Taktperiode
8    D_init: std_logic_vector(7 downto 0) := x"3A");
9   port(
10   cs, scl:  in std_logic;
11   sda     : out std_logic);
12 end entity;
13 architecture s of ADC081S021 is
14  constant tAQC:  time := 350 ns; -- Minimum time required for acquisition
15  constant tQUIET:time :=  50 ns; -- Quiet time
16  constant tCS:   time :=  10 ns; -- Minimum CS pulse width
17  constant tSU:   time :=  10 ns; -- CS setup time prior to SCL falling edge
18  constant tEN:   time :=  20 ns; --Delay from CS until SDA TRI-STATE disabled
19  constant tCH:   time := 0.4*tSCL;-- SCL low pulse width
20  constant tCL:   time := 0.4*tSCL;--SCL high pulse width
21  constant tACC:  time :=  40 ns; --Data access time after SCL falling edge
22  constant tH:    time :=   7 ns; -- SCL to data valid hold time
23  constant tDIS:  time :=  25 ns; -- SCL falling edge to SDA high impedance
24  signal sidx: natural range 1 to 16;
25  signal D: std_logic_vector(7 downto 0):=D_init;
26 begin
27  process
28   variable tQUIET0, tAQC0, tCS0, tCS1, tSCL0, tSCL1: time := 0 ns;
29  begin
30   wait for 350 ns;            -- Wartezeit bis Sensor bereit
31   loop
32    wait until cs='0'; tCS0 := now;
33    assert now-tCS1 > tCS report "tCS=" & integer'image((now-tCS1)/1 ns)
34           & "ns: Minimum CS pulse width violation ";
35    assert (now-tAQC0)   > tAQC    report "tAQC=" & integer'image((now-tAQC0)/1 ns)
36           & "ns: Minimum time required for acquisition violation";
37    assert (now-tQUIET0) > tQUIET  report "tQuiet="
38           & integer'image((now-tQUIET0)/1 ns) & "ns: Quiet time violation";
```

```
39      sda <= '0' after tEN;
40      for idx in 1 to 16 loop
41       wait until scl = '0'; tSCL0:= now;
42       if idx=0 then
43        assert now-tCS1 > tSU report "tSU=" & integer'image((now-tCS1)/1 ns)
44             & ": CS setup time prior to SCL falling edge violation";
45       else
46        assert now - tSCL1 > tCH report "tCH=" & integer'image((now - tSCL1)/1 ns)
47             & "ns: SCL high pulse width violation";
48       end if;
49       if idx>3 and idx<12 then sda <= D(11-idx) after tACC;
50       elsif idx=12 then sda <= '0' after tACC;
51       elsif idx=16 then
52        sda <= 'Z' after tDIS;
53        tQUIET0 := now + tDIS;
54       end if;
55       if idx=14 then tAQC0:= now; end if;
56       wait until scl = '1'; tSCL1:= now;
57       assert now - tSCL0 > tCL report "tCL=" & integer'image((now - tSCL0)/1 ns)
58             & "ns: SCL low pulse width violation";
59           sidx <= idx;
60      end loop;
61      if cs='0' then wait until cs='1';  tCS1:=now;
62      else tCS1 := cs'last_event; end if;
63      D <= D(6 downto 0) & (D(7) xor D(5) xor D(4) xor D(3));
64    end loop;
65   end process;
66  end architecture;
```

Alle Zeitvorgaben sind zu Beginn als Konstanten mit denselben Namen und Werten wie im Datenblatt vereinbart. Die Simulation wartet jeweils auf die Änderungen von »SCL« und »CS« und speichert zu den Änderungszeitpunkten die aktuellen Simulationszeiten[2] in Variablen. Verzögerungen werden mit »after« nachgebildet (siehe EDS, Foliensatz 2). Die Kontrolle zu zusichernder Zeitbedingungen erfolgt mit:

```
assert (<Zeitbedingung> report <Ausgabetext>;
```

Diese Assert-Anweisungen erzeugen bei Verletzung der Zeitbedingungen Ausgaben auf der Konsole aus »Dateiname«, »Zeilennummer«, »aktuelle Simulationszeit« und dem Ausgabetext. Die Ausdrücke der Form

```
integer'image((now - tSCLK0)/1 ns)
```

hinter den Reportanweisungen erzeugen eine Textdarstellung des ganzzahligen Wertes der Zeitdifferenz, im Beispiel der aktuellen Simulationszeit abzüglich der Zeit, zu der »SCL« zuvor nach null gewechselt hat, in Nanosekunden. Der Konkatenationsoperatoren »&« fügen die Teilzeichenketten incl. der Textdarstellungen der Zahlewerte zu einem Gesamttext zusammen.

Für die in der Simulation zu übertragenden Bytes (Helligkeitswerte) ist der Generic-Parameter »D_init« mit dem Standardwert 0x3A der Anfangswert. Die Datenwerte für die Folgetransfers berechnen sich nach dem Pseudo-Zufallsalgorithmus:

$$D_i = \begin{cases} D_{i-1} & \text{für } i \in \{1,2,3,4,5,6,7\} \\ D_7 \oplus D_5 \oplus D_4 \oplus D_3 & \text{für } i = 0 \end{cases} \quad (1)$$

[2] Die aktuelle Simulationszeit ist der Rückgabewert der Funktion »now«.

Aufgabe 1.1: Simulationsmodell kontrollieren

Kontrollieren Sie anhand des Datenblatts des ADC081S021, dass das Simulationsmodell das Busprotokoll richtig beschreibt. Nummerieren Sie in Abb. 1 b die eingezeichneten Verzögerungszeiten und die zu kontrollierenden Zeitbedingungen und schreiben Sie die jeweilige(n) Nummer(n) hinter die Programmzeile(n) des Simulationsmodells, die das symbolisierte Zeitverhalten im Bild nachbilden.

Aufgabe 1.2: Bestimmung der transferierten Byte-Werte

Bestimmen Sie mit Hilfe von Gl. 1 die Bytewert D_n für die Transfers 1 bis 8, wenn bei der Instanziierung wie in der nachfolgenden Tabelle »D_init« der Wert x"4D"« zugeordnet wird:

n	D_n	n	D_n
0	"01001101" (x"4D")	4	
1		5	
2		6	
3		7	

Einfachste Lösung ist eine Report-Anweisung nach der Programmzeile, in der D berechnet wird. Konvertierung von std_logic_vector in eine Textausgabe siehe EDS, Hausübung 3.

3 Simulationsmodell für den zu entwickelnden SPI-Master

Der SPI-Master hat außer den SPI-Signalen »CS«, »SCK« und »SDI« den 100MHz-Takt der Baugruppe, ein Startsignal als Eingabe und eine Startquittierung »start_ack« sowie die empfangenen 8-Bit-Daten als Ausgange. Ein erster Prozess erzeugt aus dem Baugruppentakt einen 2 MHz-SPI-Takt, der laut Datenblatt im Bereich von 1 MHz bis 4 MHz liegen und ein Tastverhältnis zwischen 40% und 60% haben soll. Das Ausgabesignal »CS« ist gleichzeitig der Automatenzustand[3]. Für Im Zustand »CS aktiv« erfolgt für 16 Takte von »SCK« die Datenübertragung. Bei steigender SCK-Flanke werden in den Takten 3 bis 10 in absteigender Reihenfolge die Bitwerte von »SDI« in das Ausgaberegister »D« übernommen. Im Zustand »CS inaktiv« wird für mindestens drei SCK-Taktperioden gewartet, um die Zeitbedingungen für t_{CS}, t_{ACQ} und t_{QUIET} einzuhalten (vergl. Abb. 1). Anschließend wird mit »start='1' and start_ack='0'« die nächste Übertragung gestartet und mit »start_ack='1'«. bestätigt. Das Quittungssignal »start_ack« ist ein Handshake-Signal, dass der übergeordneten Schaltung, in die der SPI-Master eingebettet wird, bestätigt, dass der SPI-Master die Aktivierung und Freigabe des Startsignals erkannt hat. Die Initialisierung des Automatenzustands erfolgt im Simulationsmodell durch die Anfangswertzuweisungen bei den Signalvereinbarungen für »cs« und »idx«, statt mit dem üblichen Initialisierungssignal.

```
1  ------------------------------------------------------------------------
2  -- spi_master.vhd
3  ------------------------------------------------------------------------
4  library IEEE; use IEEE.STD_LOGIC_1164.ALL;
5  entity spi_master is
6   port (t: in  std_logic;
7     start: in  std_logic;
8     start_ack: buffer std_logic;          -- Ausgabe, die auch gelesen wird
9     CS  : buffer std_logic := '1';        -- SPI Chip Select, Automatenzustand
10    SDI : in     std_logic;               -- SPI serielle Eingabedaten
11    SCK : buffer std_logic;               -- SPI Takt
12    D: out std_logic_vector(7 downto 0));-- empfangenes Datenbyte
13 end entity;
```

[3] Ausgabesignale, deren Werte innerhalb der Beschreibung gelesen werden, sind statt als »out« mit Signalflussrichtung »buffer« zu vereinbaren.

```
14 architecture a of spi_master is
15  signal idx: natural range 0 to 15 := 0;-- Bitzaehler
16  signal start_del: std_logic;           -- abgetastetes Startsignal
17  signal ct: natural range 0 to 31;      -- Zaehler fuer Taktteiler
18 begin
19  process(t) begin                       -- Erzeugung des Takts SCK mit der
20   if rising_edge(t) then
21    if SCK = '0' then                    -- Frequenz 100MHz/50 = 2MHz
22     if ct < 25 then ct <= ct +1;
23     else ct <= 0; SCK <= '1';
24     end if;
25    else
26     if ct < 25 then ct <= ct +1;
27     else ct <= 0; SCK <= '0';
28     end if;
29    end if;
30   end if;
31  end process;
32  process(SCK)                           -- Empfangsprozess mit SPI-Takt
33  begin
34   if rising_edge(SCK) then
35    start_del <= start;                  -- Abtasten des Startsignals
36    if start_del='0' then start_ack <= '0'; end if;
37                                         -- Ruecknahme des Quittungssignals
38    if CS='0' then                       -- Automatenzustand "CS aktiv"
39     if idx>2 and idx < 11 then D(10 - idx) <= SDI; end if;
40     if idx = 15 then CS <= '1'; idx <= 0;
41      else idx <= idx +1;                -- Bitzaehler erhoehen
42     end if;
43    else                                 -- Automatenzustand "sonst"
44     CS <= '1';                          -- CS inaktiv / Uebertragungspause
45      if idx < 3 then idx <= idx +1;     -- Warte mind. 3 Bitzeiten
46     elsif (start_del='1') and (start_ack = '0') then
47      CS <='0'; start_ack <='1'; idx <= 0;--Wechsel in Zustand "Uebertragung"
48     end if;
49    end if;
50   end if;
51  end process;
52 end architecture;
```

Aufgabe 1.3: Pulsbreitenvariation des generierten SPI-Taktes

Welche Einschaltdauer, welche Ausschaltdauer und welche relative Pulsbreite (Verhältnis aus Einschaltdauer und Periodendauer) hat der generierte SPI-Takt? Mit welcher Programmänderung lässt sich überprüfen, dass die Assert-Anweisungen im SPI-Slave-Modell unzulässige relative Pulsbreiten kleiner 40% und größer 60% erkennen?

Aufgabe 1.4: Operationsablaufgraph

Zeichnen Sie in Anlehnung an Foliensatz 4 (Rechenwerke und Operationsabläufe) der Vorlesung Entwurf digitaler Schaltungen den Operationsablaufgraph, den das Simulationsmodell nachbildet.

4 Testbench

Die nachfolgende Testbench verbindet die Simulationsmodelle des ADUs und des SPI-Masters und erzeugt das Start sowie das Taktsignal. Das Startsignal darf nur aktiviert werden, wenn das Bestätigungssignal inaktiv ist und wird immer kurze Zeit nach Bestätigung wieder deaktiviert (Handshake-Prinzip). Die Anzahl der simulierten Transfers wird durch die Simulationsdauer »tSIM« begrenzt. Abb. 2 zeigt das Simulationsergebnis.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity TB_ALS is end entity;
4  architecture a of TB_ALS is
5   signal start, t: std_logic := '0';
6   signal start_ack, SD, SCK, CS: std_logic;
7   constant tSIM: delay_length := 35 us;
8   constant tP: delay_length := 500 ns;
9  begin
10  slave: entity work.ADC081S021 generic map(tSCL => tP, D_init => x"4D")
11                                port map(scl => sck, sda => sd, cs=>cs);
12  master: entity work.spi_master port map(t=>t, start=>start,
13            start_ack=>start_ack, cs=>cs, sck=>sck, sdi=>sd);
14  -- geaenderter Entity fuer Post-Place & Route Simulation
15  --master: entity work.als port map(t=>t, start=>start,
16            start_ack=>start_ack, cs=>cs, sck=>sck, sdi=>sd);
17  -- process begin
18  -- wait for 0.25*tP; t <= '1';
19  -- wait for 0.75*tP; t <= '0';
20  -- if now > tSIM then wait; end if;
21  -- end process;
22  t <= not t after tP/100 when (now < tSIM) else '0';
23  process begin
24   wait for 0.5 us;
25   start <= '1' after 200 ns;
26   if start_ack /= '1' then wait until start_ack = '1'; end if;
27   start <='0' after 200 ns;
28   if start_ack /= '0' then wait until start_ack = '0'; end if;
29   wait for 9 us;
30   if now > tSIM then wait; end if;
31  end process;
32 end architecture;
```

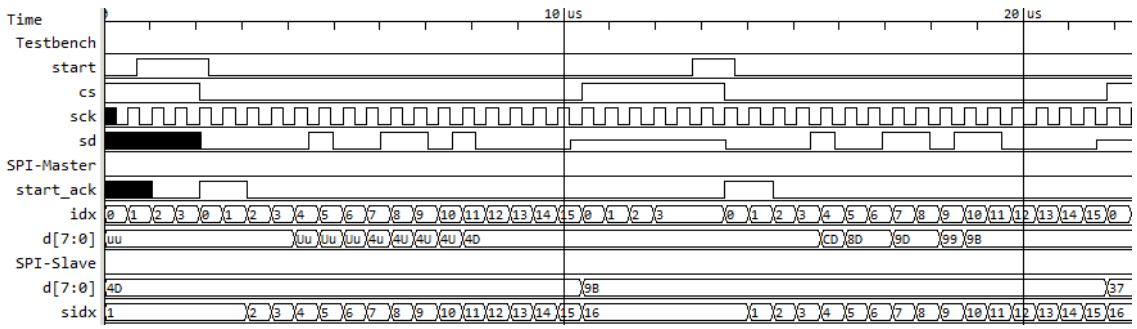

Abbildung 2: Simulationsergebnis

Aufgabe 1.5: Simulation mit ghdl

Laden und entpacken Sie die drei Simulationsdateien auf ihrem PC in ein leeres Verzeichnis. Öffnen Sie eine Konsole (Windows-Taste + R > »cmd«) und wechseln Sie in das Verzeichnis. Kommandos für das Übersetzen und die Ausführung der Simulation:

```
ghdl -a ADC81S021.vhd            # Analysieren und Vorübersetzen
ghdl -a spi_master.vhd           #  der Quellen
ghdl -a TB_ALS.vhd
ghdl -m TB_ALS                   # make (Simulationsmodell bauen)
ghdl -r TB_ALS --wave=TB_ALS.ghw # Simulation ausführen
gtkwave TB_ALS.ghw TB_ALS.sav    # berechnete Signalverläufe anzeigen
```

Passen Sie die Darstellung in gtkwave an die in Abb. 2 an und speichern Sie die Einstellungen im »Save-File«. Nach einer Änderung in einer der Quelldateien genügt die Wiederholung der Schritte ab »make«.

Aufgabe 1.6: Kontrolle der Zeitkontrollen

Ändern Sie das Simulationsmodell des SPI-Masters so, dass die Warnung

1. »... Minimum time required for acquisition violation«,
2. » ... SCLK low pulse width violation«

auf die Konsole ausgegeben wird.

Aufgabe 1.7: Kontrolle der ersten 8 Transfers

Erhöhen Sie die Simulationszeit so weit, dass nacheinander acht Datenbytes vom PmodALS zum SPI-Master übertragen werden. Vergleichen Sie die übertragenen Datenbytes mit denen in Aufgabe 1.2 bestimmten Datenbytes.

5 Implementierung

Stecken Sie das PmodALS wie in Abb. 3 links an JB des Nexys3-Boards und impementieren Sie den SPI-Master in den FPGA auf dem Nexys3-Board.

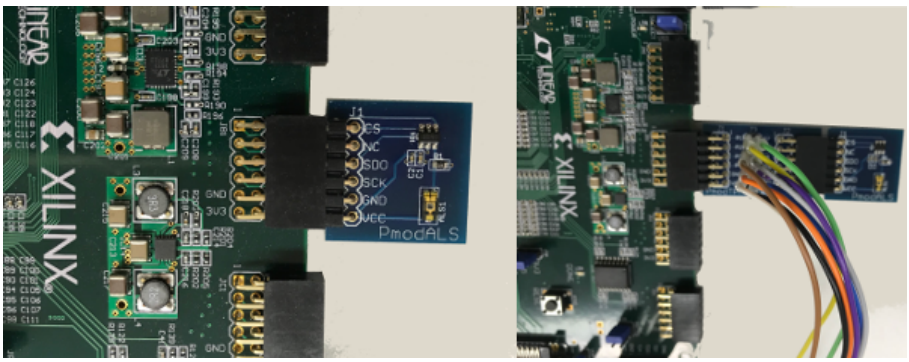

Abbildung 3: PmodALS an JB, links ohne und rechts mit angeschlossenem USB-Logi

Der SPI-Master soll folgend Anschlussignale erhalten:

- 100MHz-Board-Takt an T,

- Taster BTNU als Startsignal,
- die 8 Leuchtdioden auf dem Board für die vom Sensor empfangenen Helligkeitswerte D und
- die SPI-Signale zur Verbindung mit dem PmodALS an JB.

Die komplette ucf-Datei »ALS.ucf« finden Sie auf der Web-Seite. Abb. 4 zeigt die unter ISE einzurichtende Hierarchie.

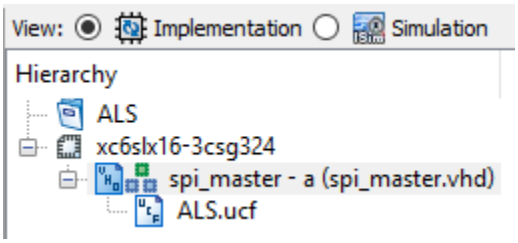

Abbildung 4: Objekthierarchie für die Implementierung

Aufgabe 1.8: Implementierung und Test

Erzeugen Sie ein neues Projekt mit den üblichen Einstellungen. Binden Sie die VHDL-Beschreibung des SPI-Masters und die ucf-Datei ein. Übersetzen, Laden und Testen Sie, dass die Ausgabe vom Lichteinfall auf den Sensor abhängt.

6 Post-Place & Route Simulation

Die Simulation mit ghdl in Aufgabe 1.5 simuliert nur das Verhalten des SPI-Masters ohne Verzögerungen. Die Kontrolle, dass die im SPI-Slave-Modell zu zusichernden Zeitbedingungen auch für die fertige Schaltung eingehalten werden, erfordert eine Post-Place & Route Simulation. Das Simulationsmodell wird in ISE in der Ansicht »Implementation« (Abb. 4) erzeugt:

- Auswahl der Beschreibung des SPI-Masters.
- Implement Design > Place & Route > Generate Post-Place & Route Simulation Model.

Für die Simulation unter ISE ist nach der Erzeugung des Simulationsmodells wie in Abb. 5 ganz oben »Simulation« und im Auswahlfeld darunter »Post-Route« auszuwählen. Der Hierarchie-Baum für die Simulation hat als oberste Einheit den Testrahmen[4] sowie das generierte »timesim«-Modell des SPI-Masters[5]. Die Datei mit dem neuen SPI-Master im Editor öffnen, geänderten Entity-Namen ablesen und Entity-Name in der Testbench anpassen. Das Hierarchie-Fenster muss danach wie in Abb. 5 aussehen. Durchführung der Simulation:

- Im Hierarchie-Fenster Auswahl der Testbench.
- Im Process-Fenster Auswahl »Simulate Post-Place & Route Model«.
- Simulate > Run All (F5).

Wie in Abb. 6 werden dann die Signalverläufe der in der Testbench definierten Signale (und Konstanten) dargestellt. Nach dem Hinzufügen weiterer Signale ist ein »Re-Launch« erforderlich.

[4] Einzufügen mit »Add Source« > Associate in Simulation ändern.
[5] Zu suchen in <Projektverzeichnis>\netgen\par*.vhd. Zu erkennen auch am Zeitstempel der Dateierzeugung.

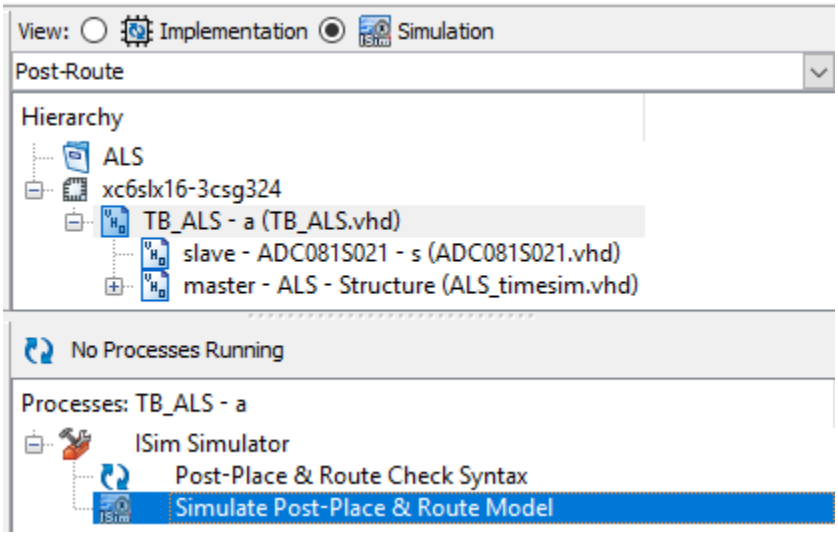

Abbildung 5: Objekthierarchie für die Simulation

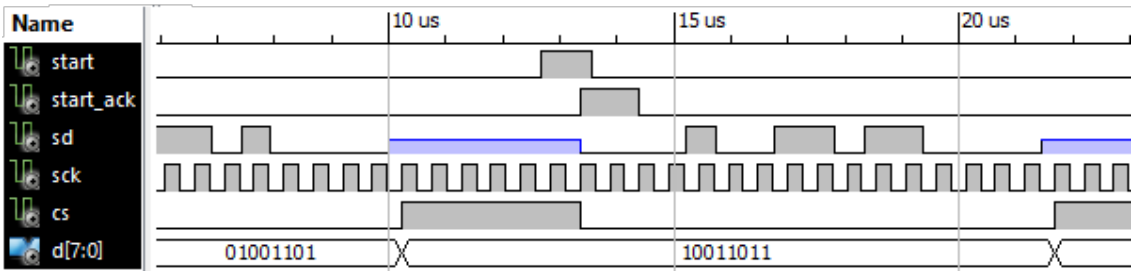

Abbildung 6: Ergebnis der Post-Place & Route Simulation

Aufgabe 1.9: Post-Place & Route Simulation

Führen Sie die Post-Place & Route Simulation durch und nehmen Sie die Einstellungen im Simulator so vor, dass die Anzeige etwa der in Abb. 6 entspricht. Speichern Sie die Einstellungen.

7 Kontrolle mit Logikanalysator

Auch bei der Implementierung können Fehler wirksam werden, die bei der Übersetzung und den Simulationen nicht sichtbar geworden sind. Deshalb sollen die Verläufe der drei SPI-Signale auch beim richtigen Betrieb mit dem USB-Logi kontrolliert werden. Dazu ist zwischen Stecker JB ein PmodTPH2 einzufügen. Die drei zu beobachtenden SPI-Signal und Masse sind wie folgt über die Steckkontakte mit LA-Eingängen zu verbinden:

Signal	CS	NC*	SDO	SCK	GND
Stecker	P1	P2	P3	P4	GND
LA-Anschluss	0	1	2	3	GND

(*– Not Connected). Bis auf die ausgegebenen Daten, die von der Beleuchtung des Sensors abhängen, sollten die Signalverläufe mit denen aus der Post-Place & Route Simulation übereinstimmen (Abb. 7).

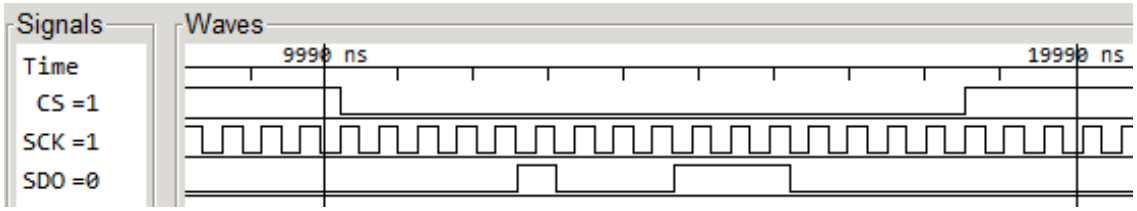

Abbildung 7: Ergebnis LA-Aufzeichnung

Aufgabe 1.10: Vergleich Logikanalyse und Simulation

Erzeugen Sie ein Bildschirmfoto von dem mit den USB-Logi aufgezeichneten Signalverläufen für den Transfer eines einzelnen Helligkeits-Byte-Wertes. Bestimmen Sie aus dem Bild den ausgegebenen Bytewert. Wiederholen Sie die Post-Place & Route Simulation mit genau diesem Wert und vergleichen Sie die Signalverläufe.

Aufgabe 1.11: Schaltungserweiterung

Der vorgegebene SPI-Master ist nur ein Basismodell mit minimalem Funktionsumfang. Denkbare Erweiterungen unter Verwendung von Entwurfsbausteinen aus dem Praktikum Teil 1 sind:

- Zusätzliches Initialisierungssignal, mit angeschlossenem Taster.
- Anzeige der Lichtwerte als 2-stellige Hexadezimalzahl auf der 7-Segmentanzeige.
- Anzeige der Lichtwerte als 3-stellige Dezimalzahl auf der 7-Segmentanzeige.
- Versenden der Lichtwerte über eine serielle Schnittstelle an den PC. Steuerung und Empfang mit HTerm.

8 Abnahmekriterien

- Aufgabe 1.1: Nummerierung der Zeitbedingungen in Abb. 1 b und Zuordnung zu den Zeilen der Datei »ADC081S021.vhd«.
- Aufgabe 1.2: Ausgefüllte Tabelle mit den transferierten Bytes.
- Aufgabe 1.3: Erforderliche Änderungen.
- Aufgabe 1.4: Operationsablaufgraph.
- Aufgabe 1.5: Die ghw- und die sav-Dateien zur Darstellung der Signalverläufe.
- Aufgabe 1.6: Einkommentierbare Änderungen, mit denen bei »run« die geforderten Warnungen auf die Konsole ausgegeben werden.
- Aufgabe 1.7: Mit gtkwave vorführbare ghw- und sav-Datei.
- Aufgabe 1.8: Ladbares Bit-File zur Vorführung der Funktion.
- Aufgabe 1.9: Projekt mit allen Dateien, um die Simulation zu starten und die gewünschte Anzeige zu erhalten.
- Aufgabe 1.10: Bildschirmfoto der Aufzeichnung des Logikanalysators und die einkommentierbare Änderung in der Testbench, mit der die Übertragung desselben Bytewertes simuliert wird.
- Aufgabe 1.11: Vorführbare Schaltung mit Initialisierungstaste und einer Form der 7-Segmentanzeige der Helligkeitswerte (hexadezimal, dezimal, ...).