FH-OÖ Hagenberg/ESD
Metrikorientierter Hardwareentwurf, WS 2015
Eugen Pfann, modifiziert nach Rainer Findenig, Markus Schutti © 2008

1. Übung: PROL16 auf dem DE1

Name(n): Punkte:

In dieser Übung sollen Sie Ihren PROL16-Entwurf für eine FPGA Implementierung auf dem DE1 modifizieren, wobei die internen Block-RAMs des FPGAs verwendet werden sollen. Ein entsprechender Assembler, der eine fertige Block-RAM-Beschreibung erzeugt, ist in `/eda/mhe3/bin` verfügbar. Er muss mit den Parametern

```
assembler.rb -i infile.asm -o outfile.vhd
```

gestartet werden. Beachten Sie, dass Sie zur Ausführung des Assemblers einen funktionierenden Ruby-Interpreter (http://www.ruby-lang.org) benötigen. Außerdem muss die Speichergröße in der *.vhd Datei auf die am FPGA verfügbare Größe reduziert werden.

1 Peripherie

Entwickeln Sie zwei Peripherieeinheiten für den PROL16 die auf verschiedenen Adressbereichen angesprochen werden können:

- General Purpose Input-Port (GPI), zum Einlesen der zehn Schalterwerte (mit Spike Filter)
- General Purpose Output-Port (GPO) zur Ansteuerung der vier 7-Segment Anzeigen

Die Busschnittstelle dieser Komponenten können Sie wie jene des erzeugten RAMs gestalten.
Zur Anbindung an die CPU benötigen Sie zusätzlich einen einfachen Adressdecoder (Abbildung 1).

Hinweis zur Speicheranbindung

Der PROL16 verwendet eine asynchrone Speicherschnittstelle, an der nun ein On-Chip-Bus verwendet werden soll. Diese Schnittstelle erwartet beim Lesen nach dem Anlegen der Adressen und Strobes noch im gleichen Zyklus die entsprechenden Daten.

Mit den in den FPGAs verfügbaren Block-RAMs ist dies nicht möglich, da entweder an den Adresseingängen oder den Datenausgängen eine Registerbank modelliert werden muss [Alt08]. Die Verknüpfung der Ausgänge `mem_oe_no` und `mem_wr_no` mit dem negierten Takt wird dann nicht mehr benötigt. Für einen On-Chip-Bus ist es auch sinnvoll, aus den Speicherzugriffssignalen einen *Read-* und einen *Write-Strobe* zu erzeugen.

2 Simulation

Erstellen Sie ein Assemblerprogram, das die 10 Schalterstellungen auf dem DE1 als binär kodierte Zahl interpretiert (Zahlenbereich 0 bis 1023) und den entsprechenden dezimalen Wert auf den vier 7-Segment Anzeigen ausgibt.

Entwickeln Sie eine Testbench mit der Sie die Funktion Ihres Designs verifizieren können.
Erzeugen Sie eine *.sof Datei und lassen Sie Ihr Design auf dem DE1 laufen.

Literatur

[Alt08] Altera. *Quartus II Version 8.0 Handbook, Volume 1: Design and Synthesis, 2008*.

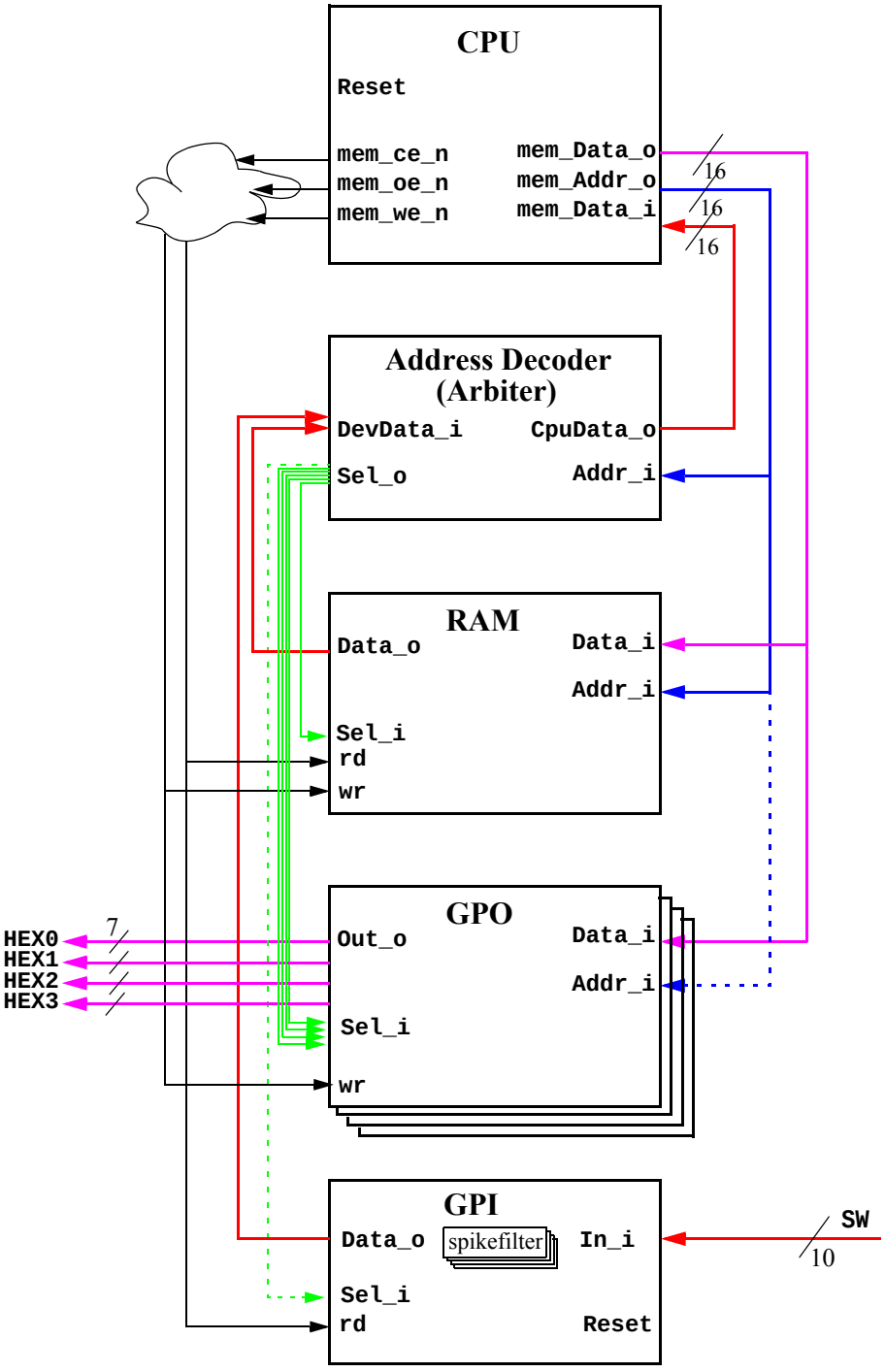

Abbildung 1: Übersicht der benötigten Einheiten und der On-Chip-Busse

PROL16 auf dem DE1
Übungsprotokoll zur Übung 1
Metrikorientierter Hardwareentwurf
Bernhard Selymes, Reinhard Penn
11.11.2015

1 Lösungsweg

Das Assemblerprogramm wandelt die Binärwerte der Schalter in eine entsprechende BCD Darstellung um. Die Schalterwerte werden mithilfe von Load an der Adresse 3000 eingelesen, die BCD Werte für die Siebensegmentanzeige werden an die Adresse 3100 geschrieben. Bei der Adresse 3000 leitet der Arbiter die Signale der Schalter als Daten an die CPU weiter. Bei der Adresse 3100 leitet der Arbiter die Daten der CPU an die Siebensegmentanzeige weiter.

2 Source Code

2.1 Anpassung der Zuweisung der Memory Zugriffsignale in der CPU

../../src/prol16/cpu.vhd

```
108   mem_ce_o <= mem_rd_stb or mem_wr_stb;
109   mem_oe_o <= mem_rd_stb;
110   mem_we_o <= mem_wr_stb;
```

2.2 Auflösung der Adresssignale im Arbiter

../../src/arbiter.vhd

```
25      if (to_integer(unsigned(addr_i)) = 3000) then
26        -- read switches
27       cpu_data_o(9 downto 0) <= gpi_data_i;
28      elsif (to_integer(unsigned(addr_i)) = 3100) then
29        -- write to 7 segment
30       sel_gpo_o <= '1';
31      elsif (to_integer(unsigned(addr_i)) < 3000) then
32        -- normal ram write/read
33        sel_ram_o <= '1';
34       cpu_data_o <= ram_data_i;
35      end if;
```

2.3 Umwandlung binäre Codierung zu Siebensegment Anzeige

../../src/bcd2sevsegment.vhd

```
21      case bcd_i is
22        when "0000"=> sev_segment_o <="0111111";  -- '0'
23        when "0001"=> sev_segment_o <="0000110";  -- '1'
24        when "0010"=> sev_segment_o <="1011011";  -- '2'
25        when "0011"=> sev_segment_o <="1001111";  -- '3'
26        when "0100"=> sev_segment_o <="1100110";  -- '4'
27        when "0101"=> sev_segment_o <="1101101";  -- '5'
28        when "0110"=> sev_segment_o <="1111101";  -- '6'
29        when "0111"=> sev_segment_o <="0000111";  -- '7'
30        when "1000"=> sev_segment_o <="1111111";  -- '8'
31        when "1001"=> sev_segment_o <="1101111";  -- '9'
32        when others=> sev_segment_o <="0000000";  -- off
33     end case;
```

2.4 Assemblerprogramm: Umwandlung von binär zu BCD

../../src/convertToBCD.asm

```
1  EQU ReadAddress, 3000
2  EQU WriteAddress, 3100
3  EQU LoopMax, 10
4
5  MACRO ShiftLeft4
6     SHL R2
7     SHL R2
8     SHL R2
9     SHL R2
10 ENDM
11
12 MACRO ShiftLeft8
13    SHL R3
14    SHL R3
15    SHL R3
16    SHL R3
17    SHL R3
18    SHL R3
19    SHL R3
20    SHL R3
21 ENDM
22
23 MACRO ShiftLeft12
24    SHL R4
25    SHL R4
26    SHL R4
27    SHL R4
28    SHL R4
29    SHL R4
30    SHL R4
31    SHL R4
32    SHL R4
33    SHL R4
34    SHL R4
35    SHL R4
36 ENDM
37
38 MACRO ShiftLeft6
39    SHL R5
40    SHL R5
41    SHL R5
42    SHL R5
43    SHL R5
44    SHL R5
45 ENDM
46
47 start:
48    LOADI R0, LoopMax ; Loop index
49    LOADI R1, 0    ; 3 downto 0
50    LOADI R2, 0    ; 7 downto 4
51    LOADI R3, 0    ; 11 downto 8
52    LOADI R4, 0    ; 15 downto 12
53
54    LOADI R7, ReadAddress
55    LOAD R5, R7
56
57    ShiftLeft6
```

```
58
59 loopStart:
60    ; add 3 if column greater euqals 5
61    LOADI R7, 5
62    COMP R1, R7
63
64    LOADI R7, noAdd1
65    JUMPC R7
66
67    LOADI R7, 3
68    Add R1, R7
69 noAdd1:
70
71    LOADI R7, 5
72    COMP R2, R7
73
74    LOADI R7, noAdd2
75    JUMPC R7
76
77    LOADI R7, 3
78    Add R2, R7
79 noAdd2:
80
81    LOADI R7, 5
82    COMP R3, R7
83
84    LOADI R7, noAdd3
85    JUMPC R7
86
87    LOADI R7, 3
88    Add R3, R7
89 noAdd3:
90
91    LOADI R7, 5
92    COMP R4, R7
93
94    LOADI R7, noAdd4
95    JUMPC R7
96
97    LOADI R7, 3
98    Add R4, R7
99 noAdd4:
100   ; shift
101   SHL R5
102
103   SHLC R1
104   LOADI R7, 0FFF0h
105   ADD R7, R1
106
107   SHLC R2
108   LOADI R7, 0FFF0h
109   ADD R7, R2
110
111   SHLC R3
112   LOADI R7, 0FFF0h
113   ADD R7, R3
114
115   SHLC R4
116   LOADI R7, 0FFF0h
117   ADD R7, R4
```

```
118
119     ; delete upper bits
120     LOADI R7, 000Fh
121     AND R1, R7
122     AND R2, R7
123     AND R3, R7
124     AND R4, R7
125
126     ; check index
127     DEC R0
128     LOADI R7, end
129     JUMPZ R7
130     LOADI R7, loopStart
131     JUMP R7
132
133 end:
134     ShiftLeft4
135     ShiftLeft8
136     ShiftLeft12
137
138     ADD R1, R2
139     ADD R1, R3
140     ADD R1, R4
141
142     LOADI R7, WriteAddress
143     STORE R1, R7
144
145     LOADI R7, start
146     JUMP R7
```

3 Simulation

Es wurde eine einfache Testbench gebaut in der im Stimuli Prozess verschiedene Schalterstellungen angelegt werden können. In der Waveform können dann die interessanten Signale überprüft werden.

4 Synthese

Es wurde ein einfaches Testbed angelegt, in dem die Polung der Siebensegementanzeige umgedreht wurde, da diese Ports low active sind.

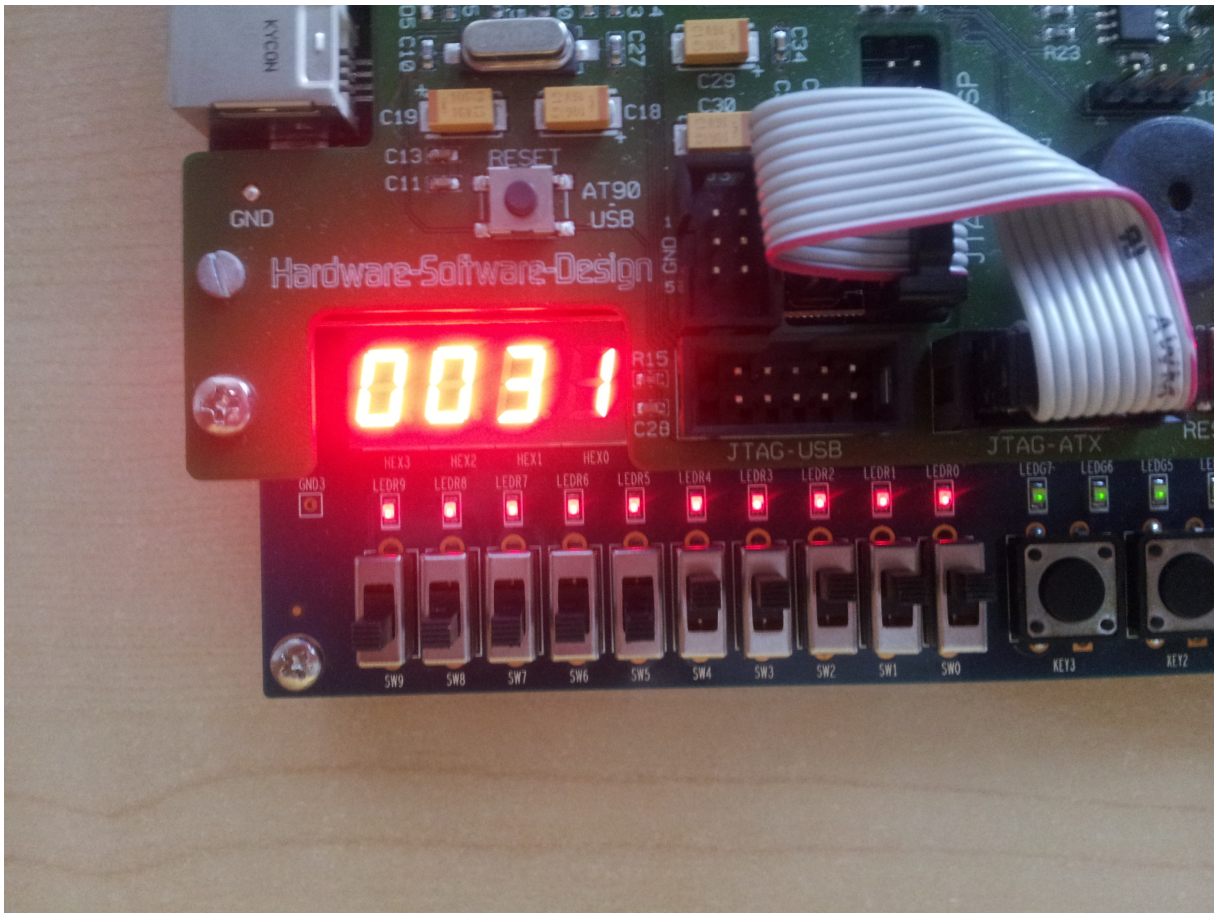

Figure 1: Test auf dem Board.