Die 7-Segmentanzeige

von Jörn Schneider

Theorie Die 7-Segmentanzeige ist die einfachste Möglichkeit die Zahlen 0 bis 9 darzustellen. Die Anzeige besteht aus 7 einzeln ansteuerbaren Leds.

Davon sind auf dem DE2-115 Board insgesamt 8 Stück vorhanden. Für die Darstellung im Binärsystem der Zahlen 0 bis 9 werden insgesamt 4 Bit benötigt. Diesen Code bezeichnetet man mit BCD (**B**inär **c**odierte **D**ezimalzahl).

1. Schritt Gestartet wird auch dieses Projekt mit dem Projekt-Wizard. Vorher wurde ein Ordner mit dem Namen Segmentanzeige angelegt (Hinweis: Die Verwendung von Sonderzeichen, Umlauten und Zahlen in den Ordnernamen und Projektnamen sollte man vermeiden, u.U. Gibt es sonst unerwartete Fehlermeldungen). Nach der Zuordnung auf den neuen Ordner wird als Projektnamen ***Segmentanzeige*** und für die ***Top-Level-Entity Seg_top*** eingetragen. Nun kann mit der Schaltfäche ***Use Existing Project Settings...*** das erste Projekt eingelesen werden. Beide Warnhinweise werden jeweils mit ***NO*** quittiert und danach mit ***Finish*** der Wizard beendet.

2. Schritt Zur Erstellung unserer VHDL-Datei verwenden wir wieder den BLOCK-Editor, den wir mit ***File New*** und der Option ***Block Diagram*** aufrufen. Nachdem wir den neuen Block erstellt haben, klicken wir mit der rechten Maustaste in den Block und in dem neuen Fenster wählen wir ***Properties*** aus (ausführlich ist dies in der Anleitung zum Flip-Flop beschrieben). Der Blockname lautet wie unsere ***Top-Level-Entity Seg_top***. Bei dem I/O-Reiter könnten wir jetzt 4 Eingabeports und 7 Ausgabeports einzeln eintragen, das ist sehr lästig. Wir tragen jetzt nur einen Eingabeport ***seg_in*** und einen Ausgabeport ***seg-out*** ein. Mit der rechten Maustaste klicken wir wieder in den Block und wählen diesmal den Punkt ***Create Designfile From Selected Block*** an. Wir markieren ***VHDL*** und als Filenamen geben wir ***Seg_top.vhd*** an. Aus dem danach angezeigten VHDL-File löschen wir wieder alle grünen Kommentarzeilen heraus. Haben wir alles richtig gemacht, erhalten wir folgendes VHDL-File:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4
5   ENTITY Seg_top IS
6
7   PORT
8   (
9   seg_in : IN STD_LOGIC;
10  seg_out : OUT STD_LOGIC
11  );
12  END Seg_top;
13
14  ARCHITECTURE Seg_top_architecture OF Seg_top IS
15
16
17  BEGIN
18
19  END Seg_top_architecture;
20
```

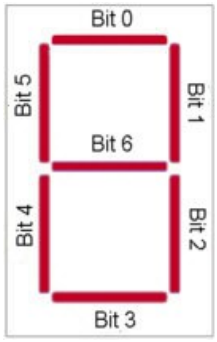
4. Schritt Nun müssen wir die Schnittstellen in der Entity per Hand ändern. Neben der Definition von einzelnen Signalen können wir auch ganze Signalblöcke, auch als Signalbusse bezeichnet, definieren. Dazu dient der Schlüsselbegriff STD_LOGIC_VECTOR. In Klammern wird dahinter angegeben, wie „breit“ der Bus ist. Für die Eingabe benötigen wir 4 Signale, für die Ausgabe 7. Möchten wir die in der Informatik übliche Nummerierung der Bits verwenden, sollten wir die Busse von links nach rechts durchzählen. Der Befehl dazu lautet z.B. (3 downto 0) für unseren Eingabe-Bus. Damit sieht unser File nun so aus:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4
5   ENTITY Seg_top IS
6
7   PORT
8   (
9   seg_in : IN STD_LOGIC_VECTOR (3 downto 0);
10  seg_out : OUT STD_LOGIC_VECTOR (6 downto 0)
11  );
12  END Seg_top;
13
14  ARCHITECTURE Seg_top_architecture OF Seg_top IS
15
16
17  BEGIN
18
19  END Seg_top_architecture;
```

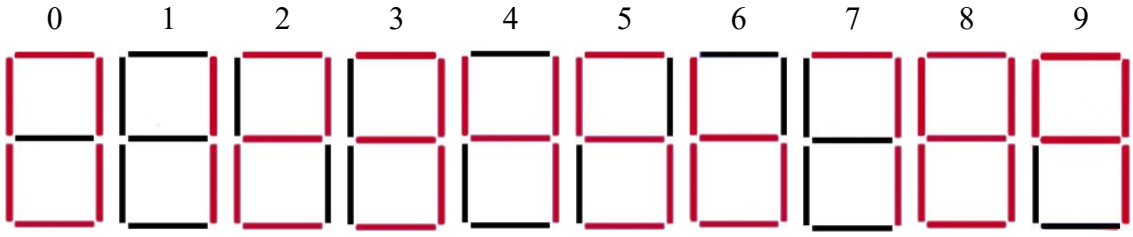
5. Schritt Nun können wir mit ***Start Analysis & Synthesis*** das Projekt für den ***Pin-Planer*** vorbereiten, den wir nach erfolgreichem Abschluss direkt aufrufen. Dort sind nun alle Pins aufgeführt (11 Stück), die wir jetzt einzeln zuweisen müssen. Die genaue Belegung aller Pins des DE2-115 ist in der PIN-Planer Anleitung zu finden. Für unsere Anzeige verwenden wir die erste 7-Segmentanzeige, die mit ***HEX0*** bezeichnet wird.

Node Name	Direction	Location	I/O Bank
in seg_in[3]	Input	PIN_R24	5
in seg_in[2]	Input	PIN_N21	6
in seg_in[1]	Input	PIN_M21	6
in seg_in[0]	Input	PIN_M23	6
out seg_out[6]	Output	PIN_H22	6
out seg_out[5]	Output	PIN_J22	6
out seg_out[4]	Output	PIN_L25	6
out seg_out[3]	Output	PIN_L26	6
out seg_out[2]	Output	PIN_E17	7
out seg_out[1]	Output	PIN_F22	7
out seg_out[0]	Output	PIN_G18	7

6. Schritt Jetzt müssen wir uns erst mal Gedanken zu unserer Anzeige machen. Wir wollen die Zahlen 0 bis 9 darstellen. Die Bits werden dabei von oben im Uhrzeigersinn von 0 bis 5 durchgezählt. Das Bit 6 ist dann der Strich in der Mitte. Das Punkt unter links brauchen wir für unsere Anzeige nicht, er ist deshalb hier nicht aufgeführt.



So sehen die einzelnen Zahlen aus:



Dies setzen wir nun in eine Wahrheitstabelle um

	BCD				7-Segment						
	Bit3	Bit2	Bit1	Bit0	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
2	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
3	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**
4	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**
5	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**
6	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
7	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**
8	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
9	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**

Wir müssen jetzt noch folgendes beachten: Die Eingabetaster liefern bei gedrücktem Taster den Wert '0' und bei nicht gedrücktem Taster den Wert '1'. Deshalb definieren wir ein internes Signal mit der Bezeichnung bcd, dies erleichtert uns später die Weiterentwicklung.

Die zweite Stolperfalle ist, dass die 7-Segmentanzeigen eine gemeinsame Anode (+ Pol) haben. Daher leuchtet bei einer logische '0' das Segment, bei einer logischen '1' nicht. Wir definieren auch hier ein internes Signal mit der Bezeichnung seg7, welches genau unserer Wahrheitstabelle entspricht.

Nun sind wir in der Lage, unsere Anzeige umzusetzen.

Hinter ARCHITECTURE werden die beiden besprochenen Signale definiert. Beide müssen die gleiche Busbreite haben wie die Eingangs- bzw. Ausgangssignale.

```
14  ARCHITECTURE Seg_top_architecture OF Seg_top IS
15
16    signal bcd : std_logic_vector(3 downto 0);
17    signal seg7 : std_logic_vector (6 downto 0);
18
19  BEGIN
```

Mit einer ***CASE***-Liste wird nun die Wahrheitstabelle umgesetzt. Die Bedingung ist das ***bcd***-Signal, dem Signal ***seg7*** wird dann genau unsere Wahrheitstabelle zugeordnet.

Mit ***seg_out <= not seg7*** erfolgt die Umsetzung auf die 7-Segmentanzeige. Würde man eine 7-Segmentanzeige mit gemeinsamer Kathode (- Pol) benutzen, könnte diese Anweisung entfallen.

Der Befehl **others** in der ***CASE***-Liste ist wichtig, da sonst undefinierte Zustände übrig bleiben, was zu einer Fehlermeldung führt. In unserem Fall wird die Anzeige bei den verbleibenden 6 Zuständen einfach dunkel geschaltet.

```
21  process (seg_in)
22
23  BEGIN
24
25  bcd <= not seg_in; -- Taster gedrückt = '0' !
26
27     case bcd is
28
29     when "0000" => seg7 <="0111111";
30     when "0001" => seg7 <="0000110";
31     when "0010" => seg7 <="1011011";
32     when "0011" => seg7 <="1001111";
33     when "0100" => seg7 <="1100110";
34     when "0101" => seg7 <="1101101";
35     when "0110" => seg7 <="1111100";
36     when "0111" => seg7 <="0000111";
37     when "1000" => seg7 <="1111111";
38     when "1001" => seg7 <="1100111";
39     when others => seg7 <="0000000";
40
41     end case;
42
43     seg_out <= not seg7; --Gemeinsame Anode '0' = Licht an!
44
45  end process;
46
47  END Seg_top_architecture;
```

Die nun folgende Übersetzung und Übertragung ist in den vorigen Projekten ausführlich erläutert worden.