Dünner Kirchweg 77
32257 Bünde
Germany
www.trenz-electronic.de

Spartan-II Development System

2002-May-4

Application Note: Buttons & Lights

Introduction

The Spartan-II Development System is designed to provide a simple yet powerful platform for FPGA development, which can be easily expanded to reflect your application's requirements.

The following application note was developed to give the engineer a quick hands-on experience, and to demonstrate the board's features and their application.

General Overview

The *TE-BL Expansion Board* provides a set of standard functionality which is commonly used by most applications:

- 7-segment displays
- LEDs
- Push buttons
- VGA output
- USB type "B" receptacle
- 48MHz oscillator

To speed-up application development, a standard VHDL module (*entity tebl*) was created, encapsulating the following functions:

- encoding of 7-segment displays
- multiplexing of LEDs
- debouncing of push buttons.
- emulation of switches
- generation of VGA timing
- encoding of USB signals

Based on the above mentioned VHDL module, this application note describes a simple design, demonstrating the following tasks:

- using the 7-segment displays
- using the LEDs
- using the push buttons
- emulating switches
- using the VGA output

Furthermore, the underlying principles are briefly explained, for reference purposes.

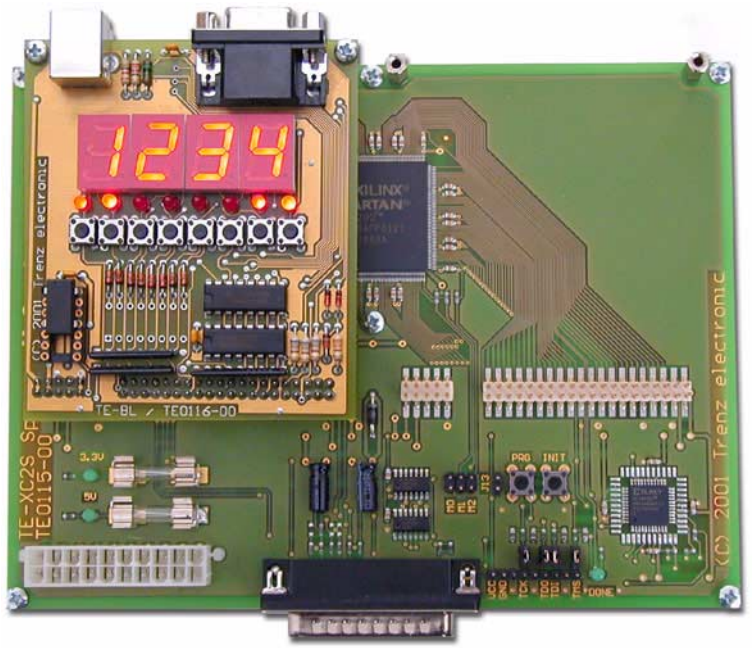

Figure 1: TE-XC2S Base Board with TE-BL Expansion.

Architectural Description

As most of the functionality is contained in the readily provided module *tebl*, only very few additional files are required to implement this application note. *Figure 2* visualizes the design hierarchy.

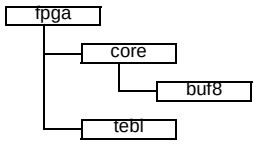

Figure 2: Design hierarchy

Entity core

The entity *core* implements the application note's specific functionality:

- display push button signals on 7-segment displays
- display switch signals on 7-segment displays and LEDs
- create test pattern on VGA output

To implement this functionality, less than 10 lines of VHDL code are required. To make things even easier to understand, we used the schematic shown in *Figure 3*.

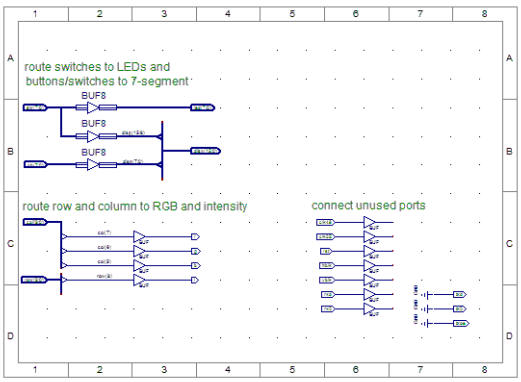

Figure 3: Entity core

push buttons

To display the state of the push buttons on the 7-segment displays, only a single line of VHDL code is required:

```
disp(7 downto 0)<= pb;
```

The entity *tebl* takes care of button de-multiplexing and de-bouncing, so that the signal *pb* may be used to control further logic directly. *Figure 4* details this functionality.

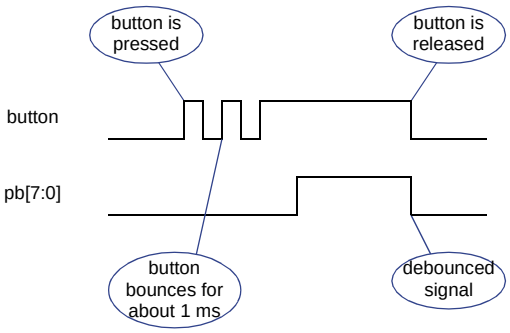

Figure 4: Debouncing push buttons

Furthermore, the entity *tebl* performs the 7-segment decoding and display multiplexing, so that the value assigned to *disp[]* is displayed in hexadecimal. See *Table 1* for further details.

disp[3:0] disp[7:4] disp[11:8] disp[15:12]	U5 U6 U7 U8	disp[3:0] disp[7:4] disp[11:8] disp[15:12]	U5 U6 U7 U8
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	b
0100	4	1100	C
0101	5	1101	d
0110	6	1110	E
0111	7	1111	F

Table 1: 7-segment decoding

switches

To emulate switches using the push buttons and display the status of the switches on the 7-segment displays and LEDs, only two lines of VHDL code are required:

```
disp(15 downto 8)<= sw;
led<= sw;
```

The entity *tebl* performs the emulation of the switching functionality, see *Figure 5* for further details.

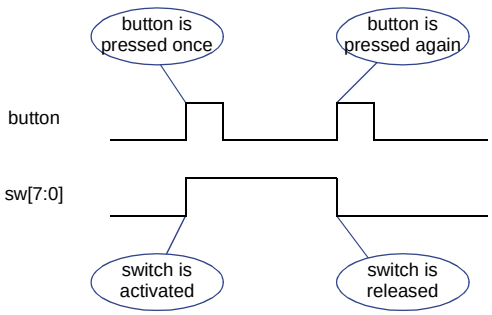

Figure 5: Emulating switches

Using schematics, things look slightly different. The *pb[]* and *sw[]* busses need to be renamed, so that they may be combined to form the *led[]* bus. To rename the signals, a symbol *buf8* was created, which is just copying its input to its output. The synthesizer will not create any logic from this symbol. See *Figure 6* for further details.

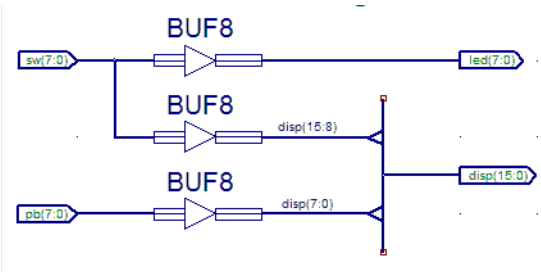

Figure 6: buttons & switches schematic

VGA test pattern

Displaying a simple test pattern on a VGA monitor requires another four lines of VHDL code:

```
vr<= vcol(8);
vg<= vcol(7);
vb<= vcol(6);
vi<= vrow(8);
```

The entity *tebl* performs the generation of the VGA timing, takes care of the blanking and provides row and column info to the application. The application in turn provides the RGB tuple plus an additional intensity signal to *tebl*. With a video resolution of 640x480 pixels, the lines above create the pattern illustrated by *Figure 7*.

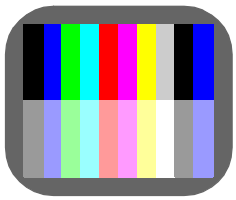

Figure 7: VGA test pattern

Again, with schematics things look slightly different. Here we used the predefined *buf* symbol to be able to rename the signals as required. See *Figure 8* for further details.

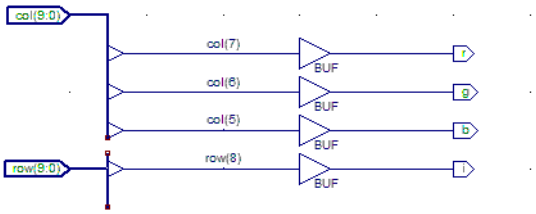

Figure 8: VGA schematic

Creating VGA images is a complex task. Therefore we created a dedicated application note for in depth coverage of this topic: The *Game of Life*.

To make sure, the schematic is properly compiled according to the design-rule-check, unused signals need to be assigned. We used some *buf* and *gnd* symbols to do so, as *Figure 9* details.

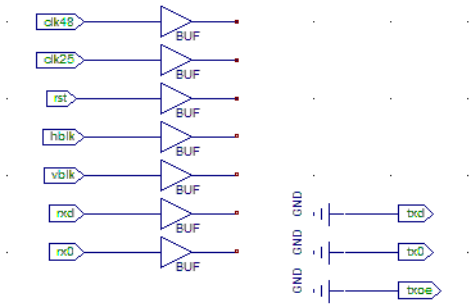

Figure 9: unused signals

Entity fpga

The entity *fpga* is the top level of the design. It performs the following functions:

- create a reset signal
- lock all I/Os to their pad locations
- specify timing constraints

While Xilinx recommends to assign design constraints using the *Constraints Editor*, this application note uses VHDL attributes to pass the constraints to the implementation tools. This is especially useful for the novice user, as the design contains less files and is more consistent. The mechanism used here was proven with Xilinx' *WebPACK ISE 4.1WP3.x*. In case your tool chain does not support this mechanism, the constraints me be additionally entered into a *.ucf* file.

Reset-on-Configuration

The following code excerpt demonstrates creation of a reset signal, which is automatically generated on FPGA configuration. It utilizes the Xilinx' standard library *UNISIM* and the *ROC* component contained herein.

```
library UNISIM;
use UNISIM.VCOMPONENTS.all;

entity FPGA is
    ...
end FPGA;

architecture XILINX of FPGA is
    signal RST : STD_LOGIC;
    ...
begin
    U2 : ROC port map(O=> rst);
    ...
end XILINX;
```

Pin locking

The following code excerpt demonstrates the specification of pad locations in a VHDL file:

```
entity FPGA is
    port(
        clk_a1 : in STD_LOGIC;
        iop_a1 : out STD_LOGIC;
        ...
    );

    attribute LOC: string;
    attribute LOC of clk_a1:
                signal is "P185";
    attribute LOC of iop_a1:
                signal is "P191";
end FPGA;
```

Notice, that the attribute is assigned within the entity declaration, not within the architecture body.

Timing constraints

Timing constraints are required to notify the implementation tools about the speed of the implemented circuits. This is important to ensure, that the design works at the desired clock speed.

The following code snippet demonstrates the specification of timing constraints within a VHDL file:

```
architecture XILINX of FPGA is
    signal    clk25 : STD_LOGIC;
    attribute PERIOD: string;
    attribute PERIOD of clk25:
            signal is "24MHz";
    ...
begin
    ...
end XILINX;
```

In addition to specifying the constraints in the VHDL code, you need to enable timing constraints in the synthesis properties. See *Figure 10* for a screenshot.

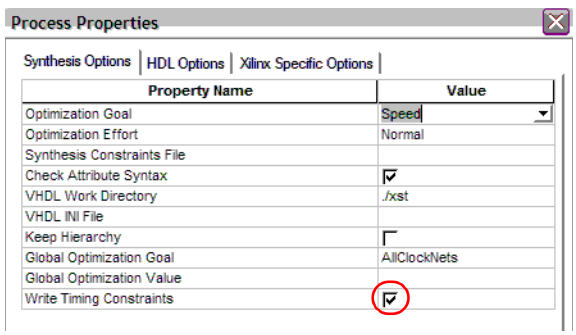

Figure 10: synthesis properties

The mechanism described here was tested with the Xilinx *XST* synthesizer. Other tools may use different mechanism to pass attributes from VHDL to the backend tools. You should check your place&route report, and verify that the constraints have been passed through the tool chain.

Entity tebl

The entity *tebl* is provided as VHDL source code and extensively commented. It is left up to the interested engineer, to review this source.

Project files

The project files for this application note are provided in *WebPACK ISE* format with all synthesis options set up to achieve a push button flow. Furthermore, the resulting *.mcs* file to program the Flash PROM and a *.bit* file to program the FPGA via JTAG are provided.

To answer questions regarding synthesis, implementation and download of projects, our *Tutorial on WebPACK ISE* is highly recommended.

References

- *Spartan-II Development System Product Specification*
  Trenz Electronic
  September 12, 2001
- *Spartan-II Development System Tutorial on WebPACK ISE*
  Trenz Electronic
  September 2001
- *Spartan-II Development System Application Note: Game of Life*
  Trenz Electronic
  September 20, 2001

Revisions History

Version	Date	Who	Description
1.0	2001sep15	FB	Created
1.1	2002may04	FB	ISE 4.2

Table 2: Revisions History