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

Spartan-II Development System

2001-October-10

Application Note: System-on-Chip with USB

Introduction

Application of IP-Cores requires in-depth knowledge of the core's behavior. Building up this know-how is greatly simplified by comprehensive reference applications. The following application note was developed to give the engineer a quick hands-on experience and a good starting point for their own developments.

General Overview

This application note describes a System-on-Chip with USB consisting of the following building blocks:

- *USB Function Controller.* The *TE-USB* Core implements the complete USB transaction layer. It is completely hardwired for speed and needs no firmware intervention. The functions of this entity include: frame recognition/ generation, parallel/ serial conversion, bit-stuffing/ de-stuffing, CRC checking/ generation, PID verification/ generation, address recognition and handshake evaluation/ generation.
- *Microcontroller & Firmware.* The *TE-51* Core in conjunction with the firmware implements the USB Device Framework (Chapter 9 Commands). Being compatible to the industry standard MCS 51 family, the TE-51 Core protects your software investment and cuts down development time.
- *Functional Block.* This functional block implements the application's unique device functionality. In this application note these are four simple 7-segment displays with registers which can be written to and read from.

To ease reuse of the USB interface circuit even for inexperienced developers, the USB Function Controller, the microcontroller and firmware are combined in a simple USB Macro.

The project is implemented on Trenz Electronic's *TE-XC2S* Platform, an FPGA Development System based on the Xilinx Spartan-II family. Besides the FPGA, this board provides all the peripheral components needed, to implement a USB Device.

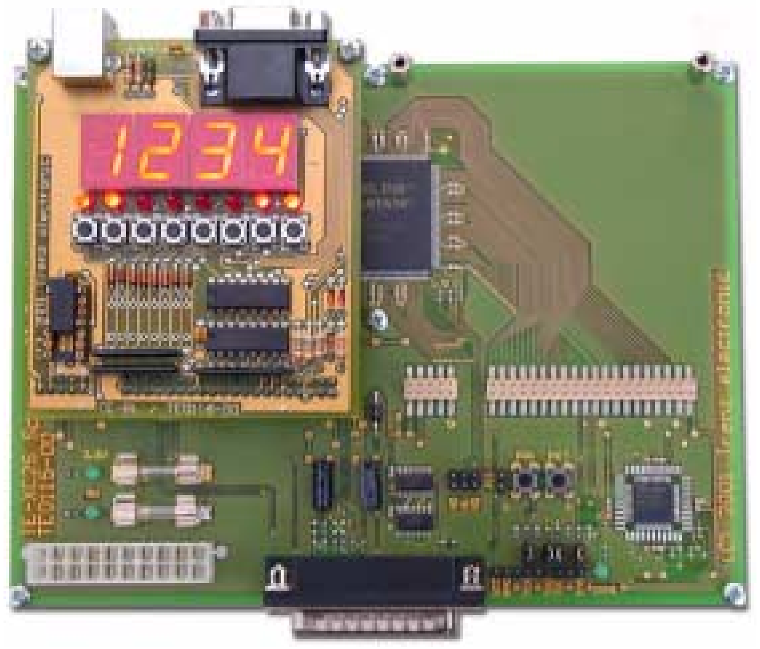

Figure 1: TE-XC2S Base Board with TE-BL Expansion

Architectural Description

The design is created from several entities which are described in the following. The design is amazingly simple, as *Figure 2* reveals. This is caused by the fact, that the complete USB functionality is encapsulated inside a simple macro, which is created from an independent project and instantiated here as a black box.

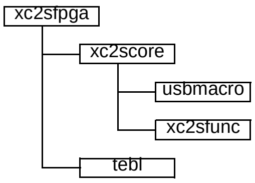

Figure 2: Design hierarchy

USB Macro

The entity *usbMacro* combines the USB Function Controller, the microcontroller plus firmware, and the glue logic into a single entity. This entity was designed to be easily reused in other simple designs. It provides the following features:

- 16-bit output
- 16-bit input
- HID-compatible firmware

This macro is provided free-of-charge under the Gnu Public License as a synthesized netlist. It combines the features of Trenz Electronic's IP Products *TE-51* and *TE-USB* and illustrates their use in a System-on-Chip environment. *Figure 3* shows the block diagram of the macro.

Entity *usbEP0* implements the USB Function Controller. A 48MHz oscillator drives *clk48. An* asynchronous, active-high reset on *rst* is required during power-on. The 12MHz clock is generated by the internal digital PLL, therefore *clk12* is driven by *clk12o*. The generics *epin_mask*, *epout_mask*, *epsetup_mask* and *episo_mask* are set up to define Endpoint Zero as the control endpoint, and Endpoint One as an IN-only Endpoint without the isochronous transfer capability. Refer to Trenz Electronic's *TE-USB* Product Specification for further details on the functionality of this IP core.

Entity *te51c* implements the MCS 51-compatible microcontroller. As this is a SoC design, Port 0 multiplexing was omitted, resulting in simplified footprint for this entity. The controller is clocked from a 24MHz clock, divided from the global clock by entity *clkdiv*. Refer to Trenz Electronic's *TE-51* Product Specification for further details on the functionality of this IP product.

Entity *xsvROM* implements the HID-compatible device firmware, which is less than 1kB in size. This entity is implemented using a *CoreGenerator* module for the highest possible implementation density.

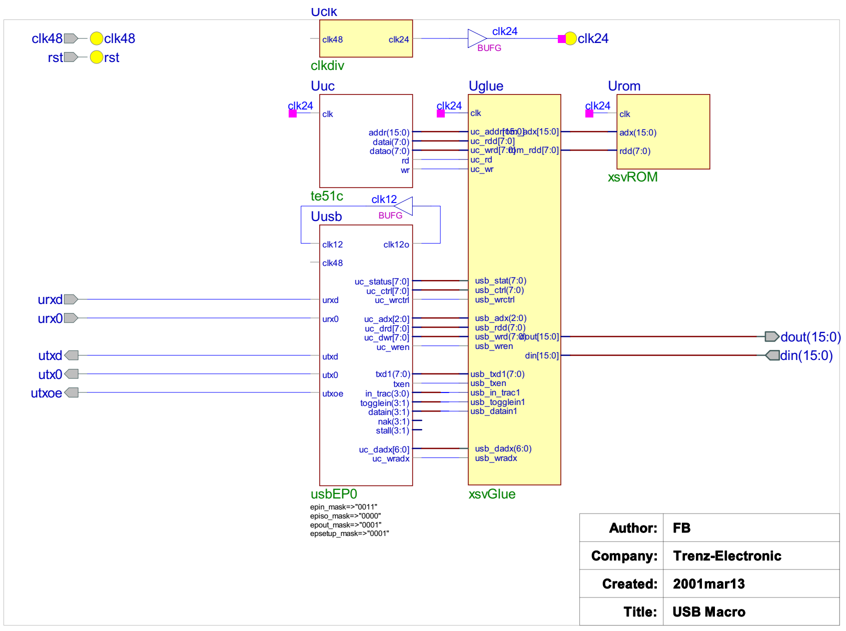

Figure 3: USB Macro

The entity *xsvGlue* implements glue logic to connect the USB Function Controller with the microcontroller. Datapathes are included to access the FIFO, the Control/Status Word and the Device Address, these circuits are memory-mapped into the 8051's address space for easy implementation. In addition to these datapathes, the endpoint control logic for the 16-bit I/Os is implemented here. This is basically two simple state machines plus output registers. 16-bit I/Os were chosen to provide increased flexibility for simple devices. In case more than 16 I/Os are required, some of the output signals may be used for addressing purposes.

Core Logic

The entity *xc2sCore* combines the USB functionality with the application-specific Functional Block to create the core logic of the design. Refer to *Figure 4* for the block diagram.

While this is a very simple application note, the concept of isolating the Functional Block from the USB interface circuitry should be kept for more complex designs. Doing so gives significant advantages during simulation and implementation, as this concept allows partial simulation and incremental synthesis efficiently cutting down development and turnaround time.

Functional Block

The entity *xc2sFunc* implements the functional block of this application note which is self-explanatory: As *Figure 5* details, the only functionality is to route the USB macro's 16-bit output to the four 7-segment displays and feed this value back to the USB macro's 16-bit input. Feel free to add more to it...

Keeping the Functional Block exceptional simple, helps qualifying the minimum design requirements in terms of FPGA resources as we will see later.

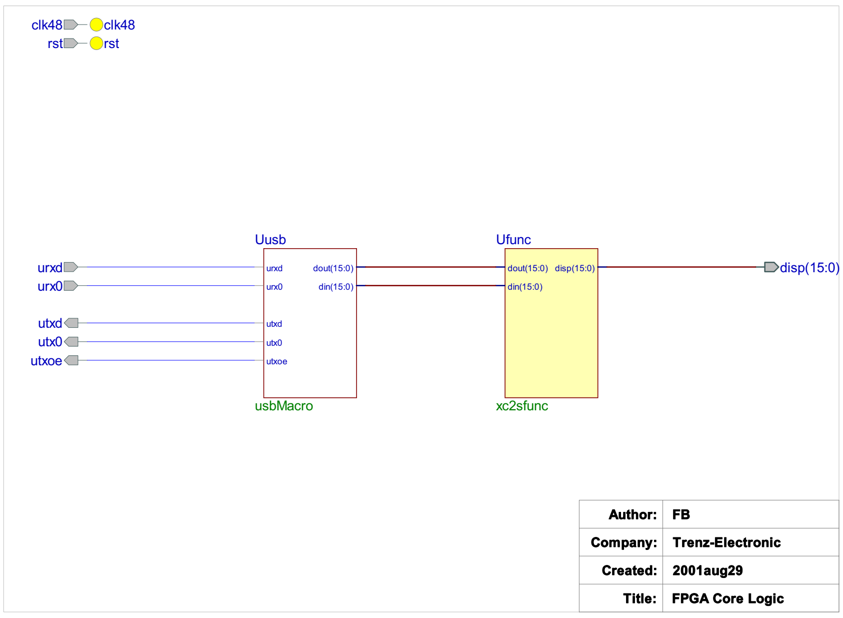

Figure 4: Core Logic

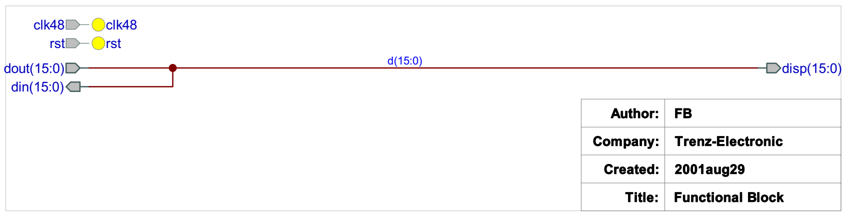

Figure 5: Functional Block

Top level

The entity *xc2sfpga* represents the top level of the design. This straightforward entity instantiates the core logic and TE-BL interface.

Furthermore, the pad locations are constrained here. For further information specifying implementation constraints via VHDL attributes, please refer to our *Buttons&Lights* application note.

Buttons & Lights interface

The entity *tebl* encapsulates the specific interface functionality required by the *Buttons&Lights* Expansion board. The main tasks of this entity are:

- led multiplexing
- 7-segment decoding
- push button debouncing
- switch emulation
- VGA timing generation
- USB transceiver interface

For further details on this entity, please refer to our *Buttons&Lights* and our *Game of Life* application notes.

Synthesis and Implementation

Synthesis and Implementation of the design were done in two steps, each with different software packages.

- The USB Macro was synthesized with Xilinx Foundation Express F3.1i, using Synopsys' FPGA-Express v3.4.
- The application note was synthesized and implemented with Xilinx *WebPACK ISE*, v3.3WP8.x.

To rebuild the USB Macro, purchase of the *TE-USB* and *TE-51* IP cores is required. For this reason, the EDIF netlist resulting from synthesis of these cores is provided with the project files.

To build the application note, make sure not to include the USB macro's EDIF netlist in the synthesis project. Instead, this entity remains unresolved during synthesis, which will force the synthesizer to insert a black box. During design implementation, *ngdbuild* resolves the black box with the macro's netlist- copy the netlist into the implementation directory to make this work.

To ensure that the design meets all timing requirements, it is required to constrain timing according to Table 1. By doing so, Place&Route is advised of fast datapathes and ensures proper timing. Even though the signals *clk12i* and *clk24i* are buried inside the entity *usbMacro*, proper setup of timing constraints is vital for error-free operation.

signal	constraint
clk_a1	PERIOD 48MHz, HIGH 50%
Ucore/Uusb/CLK12i	PERIOD 12MHz, HIGH 50%
Ucore/Uusb/CLK24i	PERIOD 24MHz, HIGH 50%

Table 1: Timing Constraints

Table 2 summarizes the resource usage of the complete design implemented in a Xilinx XC2S200 device. Please note, that these figures may vary slightly depending on the implementation tools.

Resource	Usage	
Number of Slices	804	34%
Number of Slice Flip Flops	537	11%
Total Number 4 input LUTs	1,320	28%
Number used as LUTs	1,248	
Number used for 32x1 RAMs	64	
Number used as 16x1 RAMs	8	
Number of bonded IOBs	23	16%
Number of Tbufs	40	1%
Number of Block RAMs	2	14%
Number of GCLKs	3	
Number of GCLKIOBs	1	
Total equivalent gate count	54,449	

Table 2: XC2S200 Resource Usage

USB Enumeration

When connecting the board to a USB host controller, the host will enumerate the design with the following device descriptor:

```
bcdUSB:             0x0110
bDeviceClass:         0x00
bDeviceSubClass:      0x00
bDeviceProtocol:      0x00
bMaxPacketSize0:      0x08 (8)
idVendor:           0x0BD0
idProduct:          0x0100
bcdDevice:          0x0100
iManufacturer:        0x01
0x0409: "Trenz Electronic"
iProduct:             0x02
0x0409: "TE-USB"
iSerialNumber:        0x00
bNumConfigurations:   0x01
```

This declares the device to be compatible with the *Device Class Definition for Human Interface Devices (HID)* specified by the USB Implementers Forum. Following the HID specification ensures maximum compatibility with standard operating systems, and avoids the need for custom driver development.

Host software

To allow easy verification of the USB functionality, a simple host software running on Windows is provided. After starting the software, you will be asked for the Vendor and Product ID, which should be entered in hexadecimal format:

```
VID: 0x0bd0
PID: 0x0100
```

After entering a *w*, you can write data to the device:

```
r, w, x: w
1: 0x55
2: 0xaa
hidWrite: 55 AA
```

The 7-Segment displays will light up according to the data being sent.

After entering an *r*, the software reads back data from the device:

```
r, w, x: r
hidRead: 55 AA
```

The data being read, will reflect the current status of the 7-Segment displays.

After entering an *x*, the software quits.

The host software is provided in source form “as-is” and is meant to serve as a good starting point for your own developments. Refer to the References Section for further readings.

References

- *Spartan-II Development System Product Specification*
  Trenz Electronic
  http://www.trenz-electronic.de
- *Full-Speed USB 1.1 Function Controller Product Specification*
  Trenz Electronic
  http://www.trenz-electronic.de
- *MCS 51 Compatible 8-bit Microcontroller Core Product Specification*
  Trenz Electronic
  http://www.trenz-electronic.de
- *Universal Serial Bus Specification*
  USB Implementers Forum
  http://www.usb.org
- *USB Design by Example*
  John Hyde
  Intel Press
  http://www.usb-by-example.com

Revisions History

Version	Date	Who	Description
1.0	01mar19	FB	Initial version
1.1	01may03	FB	XSV-300 version
1.2	01oct10	FB	TE-XC2S version

Table 3: Revisions History