"TouchScreen" project

**Albert Dorn, Eduard Heidt, Jan Helber,
Alexander Irro and Cedric Pilot**

Important notes for FPGA programming V0.1
written by Alexander Irro

This documentation contains some important notes for the FPGA
used in the project "TouchScreen TI6"

V0.1	2006-12-8	Initial Release

Powered by LaTeX
generated at 10:02 on December 11, 2006

Contents

1 Preamble

The board soldered by the preceding group from summer term 2006 (SS06) showed that there are some hardware failures on the board. We found out that there is a capacity problem inside the JTAG interface giving us an undefined behaviour after some JTAG transfers (around six or seven). Also the touch matrix of the LCD is soldered incorrectly because five of the eight connectors are earthed all the time so only the first row shows correct signals that can already be handled by the ER1. In the shortness of time we were unable to fix up the hardware problems so we tried to give future groups a good development base by giving good examples and statements which parts are successfully tested or still experimental so that they know where they should continue with their development.

2 Quartus 2 project

Please open up the file controller.qpf inside Altera Quartus 2 and click inside the "Project Navigator" frame on the "Files" tab and expand the directory "Device Design Files". Each entity is commented adequate and we will explain some of the entities briefly.

2.1 Entities outside ER1

2.1.1 Controller.vhd

This module is the main module including all others. Here we implemented all parts "outside" the ER1 IP-Core. There are the two data registers DatLow/DatHigh transferring new command data to the ER1. We used two registers because only 8 bits of the ARM data bus are connected to the FPGA but one instruction for the ER1 has 16 bits. Both outputs are connected to the Data_Ext_In bus going inside the ER1.
For managing the bootloader transfers there are also two control registers described more detailed in the "System Architecture Document". The registers named DatCtrlIn and DatCtrlOut are also connected to the Dat_Ext_In/Dat_Ext_Out busses going inside the ER1.
As a third important entity the Bit_Debug is in principle a data flipflop toggling the debug mode realized as a 1 bit register. The FlagDebug output signal is connected directly to the clock_divider entity.

2.1.2 AdressDecoderARM.vhd

To manage access to the registers there is an address decoder necessary. This one is for the ARM and has the address range 0x83F00000 until 0x83F00034. The signal descriptions should be self explaining.

2.1.3 Clock_divider.vhd

Clocking of the ER1 is generated in this entity. If the signal debugmode is 0 the on-board quarz oscillator is used for clocking but if the signal is going to 1 the high level of the ARM address bus is used for cycling directly so there is no separate trigger signal on the data bus necessary.

2.1.4 IOFlag.vhd

This is a 1-Bit register realised with a data flipflop. It is accessible and writable by the ARM and stores the last given Data level but only the lowest bit out of the ARM's data bus is relevant.

2.1.5 Dat8Reg.vhd

The new command data for the bootloader is stored inside the two registers DatLow/DatHigh. These entities are standard 8 bit registers wired as a chain in controller.vhd.

2.1.6 IO8Reg.vhd

In principle IO8Reg is the same as Dat8Reg but is used for controlling the bootloader transfer. It has some different chipselect signals and they are only used in the Control8RegOneWay entity. Please refer to the "System Architecture Document" chapter 6.2 to get some additional information.

2.1.7 Control8RegOneWay.vhd

This entity implements the circuit plan shown in "System Architecture Document" chapter 6.2.1 and is built out of two data flipflops, some latches and two IO8Regs. In controller.vhd there are two Control8RegOneWay entities: One for managing the transfer from the ARM to the ER1 and one for the transfer from the ER1 to the ARM to avoid arbitration situations between th ER1 and ARM.

2.1.8 Working_led.vhd

To debug your signals simply you can use this entity. As a default the LED shows the clock of the ER1 but you can use any other signal as an input to improve your debug situation.

2.2 Entities inside ER1

2.2.1 er1.vhd

This is the main entity of the Einfachstrechner. It bundles all parts (RAM, ROM, etc.) of the ER1. We didn't change much inside the ER1 except: We brought the DIN and DOUT data busses outside to manage boot loader communication. Also we changed the ROM to the architecture of a RAM and tried to implement our bootloader into a separate ROM but because of the hardware failure this unit could only be tested in theory. The circuit plan generated by the RTL netlist viewer of Quartus 2 showed correct wiring. Also the bootloader copy routine and the polling routine of the ER1 have been tested successfully inside the ER1 simulator (Java applet).

3 Known bugs, pitfalls and tips

3.1 Bad port drivers

It is important that each new entity you create which is connected to the data bus (either ARM or ER1) has a Tri-State portdriver. If not you will have a real bad debugging situation because you can expect an unpredictable behaviour of the whole system.

3.2 Shared ressources: Decoupling ARM and ER1

If you try to implement a shared component please note that you have to take care about decoupling. If you have multiple outputs driving against another (ARM versus ER1) you can seriously damage hardware components (shortcuts and other effects). The DatCtrl registers are a good example for decoupling the two address busses easily without using a bus arbitration.

3.3 ARM and ER1: Differnt address ranges

If you are new to the project you have to know that there are two address spaces. One for the ARM data bus (0x83F00000 - 0x83F00034) implemented in AdressDecoder-Arm.vhd and an independant address space inside the ER1 (000 - FFF) implemented in adrdec1.vhd. They are completely different and have nothing to do with each other.

3.4 Default setting: Unused pins as Tri-State

As a default option Quartus 2 assigns unused pins to drive to ground. This might be a problem because of a possible shortcut the FPGA could get very hot or in some cases also damaged. Please refer in Quartus 2 to "Assignments" -> "Settings" -> "Device" -> "Device and Pin Options" -> "Unused Pins" and select "As input tri-stated".