Partial reconfiguration and fault simulation on Altera Cyclone FPGA

Students project at the chair of computer architecture university freiburg

Markus Weiß

March 1, 2016

Contents

1 Software prerequisites

The software used in both projects were Quartus II Version 14.1/15.0 and QSYS on Windows 7/8.1.
Quartus II is a design software for FPGAs (field programmable gate array) of Altera. With Quartus II VHDL- files were generated. The Quartus II software can be configured to reach different design goals, e.g. time complexity, speed, optimization of compilation time or energy consumption.
QSYS is described as a tool for system integration, to safe time by generating automatic connection-logic for different components. The developed VHDL code from Quartus II software can be imported to QSYS and is processed there to connect different components on the dev kit board. QSYS has to initiate, configure and connect the hps (hard processor system) to the FPGA so that they can communicate.
In addition to the development software some driver have to be installed on the running operating system. To program the FPGA via the USB cable a special driver, Altera USB Blaster driver, needs to be installed. The installation of the Altera USB Blaster driver was not so intuitive as expected, cf. 3.5 problems. Also a driver for the USB-to-Serial connector needs to be installed.
The following software is used:

- Windows 7/8.1
- Quartus II (Version 14.1/15.0)
- QSYS
- Altera USB Blaster driver
- portable os for the dev kit Unix KERNEL
- usb-to-serial converter driver

2 Partial reconfiguration

2.1 Prerequisites

The hardware used in this part of the project was a development kit board from altera, namely "Terasic DE1-SoC Development Kit". This board consists of the following core components: an FPGA, named Altera Cyclone V (5SCEMA5F31C6N), a dual-core Cortex-A9 (HPS), 1GB DDR3 and 64MB SDRAM memory and different interfaces like USB,JTAG and I2C.
The communication interfaces are an on-board USB-Blaster II, USB 2.0 ports, UART interface, Ethernet port and extended module for GPIO (general purpose input/output) pins.
The FPGA is the mainly used component in this project and therefore here are some useful facts about it (Cyclone V):

- 85.000 logic cells on the board
- 4.450 kb embedded memory
- access to an external 64 MB static random access memory (SRAM)
- low energy consumption

The Cyclone V is configured through the USB Blaster interface, where a bit stream is loaded to the memory of the board. This bit stream is lost once the energy is off. Alternatively an in-system NOR flash memory can be loaded with this bit stream, which safes it although the power is taken off.
Another way to configure the FPGA is to use the hard processor system (hps). This is done either while the boot process, with "U-Boot" or the "preloader", or after the operating system is fully booted.
While booting, the preloader initiates the clock, is responsible for multiplexing pins, configures the main memory and loads the U-Boot. The configuration of the main memory of the hard processor system and the AXI Bridges is crucial, because these have to grant the FPGA access.
The universal boot loader initiates and tests hardware components. Also it downloads and executes application software. This boot-loader is required to initiate the boot process of the UNIX-kernel.
To get access from the FPGA to the main memory of the hps the Avalon-Memory-Mapped-Interface has to be configured. It enables efficient read- and write operations, interrupts, clocks, resets and control progresses. This Avalon-MM-Interface enables address based read- and write procedures in a master/slave connection.
To start a C++ program on the board an operation system for the hps is needed. The used operation system was a small sized portable linux system which was loaded to an sd card.

2.2 Idea and the benefit

The concept of partial reconfiguration is highly desirable for some special purposes. With this feature it would be easier to change partially designs or improve the functionality of an FPGA without interrupting its work progress. Special applications may need to change some logic on an FPGA while another part of it is not allowed to stop. For example it is crucial to establish a reliable flow of data without interrupting it while another part of the FPGA can still change its logic. Therefore this concept is desirable in the field of communication systems. A benefit of this idea is that the number of devices can be scaled down, therefore the power consumption and the cost can be reduced. The tasks that two or more FPGAs has handled before can now be handled by one FPGA due to the fact that only a small part of the FPGA is changed while the other part is still running.
The partial reconfiguration concept in this project is initialized by an internal host. The internal host starts the process of reconfiguration. For this the processor on the development kit (hps) programs the desired locig cells of the FPGA while the other part of the logic cells remains. Partial reconfiguration in combination with the fault simulation explained later, give a high speed and efficient way to detect faults in hardware.

2.3 Design flow and implementation

The focus of the partial reconfiguration is rather on logic blocks than DSP, memories, PLL, transceivers and I/O blocks. Functions in the periphery like GPIO or I/O registers can not be partially reconfigured.
For the partial reconfiguration of logic blocks two different regions in the top level design are needed. One static region, which continues its process and one region, which can be dynamically configured. To get an optical feedback of this configuration process, two led pattern are generated. One pattern, in the static region, highlights the leds to check if the FPGA is still running while another led pattern is loaded to the dynamic region which let us check if the other part of the FPGA is reconfigured.
To avoid errors in the in- and output of the gates, a wrapper region is needed. This region is generated to ensure that all personas, e.g. led pattern, have the same connection to the static region. This can be done by creating dummy ports.
For partial reconfiguration all non-global inputs of the pr regions must be freezed. Freezing means in this context to drive a '1' to the inputs, which ensures there is no contention between current values and those after partial reconfiguration. Therefore a freeze region is essential.
In this project a partial reconfiguration with an internal host is realised, which initializes the reconfiguration with a c/c++ program. For this purpose the processor has to communicate with the FPGA and with the memory. The communication between the hard processor system and the FPGA, the main memory and in- and output interfaces is a very important point.
A short description of the projects design flow is listed here:

1. planning the system for partial reconfiguration

2. identify which logic blocks to reconfigure
3. code the design in Quartus II
    - develop the personas
    - generate static and pr region
    - generate freeze and wrapper region
4. connect components with QSYS
    - assign partitions to logiclock regions
5. create necessary files to program FPGA with Quartus II
6. program FPGA via USB Blaster

Quartus II creates the logic in VHDL, while QSYS connects different system components which communicate and access each other. This informations are required to generate the "preloader" which coordinates the boot process.

2.4 Problems

Two crucial problems occurred during the work on partial reconfiguration.

1. No permission to use partial reconfiguration feature in Quartus Software
    - a new license was inquired
2. the FPGA Cyclone V does not support the feature of partial reconfiguration
    - could not be solved because a new FPGA was not affordable

After working 8-10 weeks on this topic the project has to be stopped due to the fact that partial reconfiguration is not supported by the FPGA. At this time the status quo was:

- literature survey
- set up soft- and hardware prerequisites
- create a project design flow
- creating different revisions for partial reconfiguration in Quartus II
- generate VHDL design
    - top level
    - wrapper
    - freeze region
    - static region + personas
    - partial reconfiguration region + personas

- assignments with QSYS
  - logic lock region assignments
  - pin assignments
- file creation for programming the FPGA (.pmsf, .msf, .sof, .rbf)

The process of programming the FPGA with partial reconfiguration is slightly different to the normal programming. A couple of files are needed and loaded to the FPGA in a specific order. Therefore the last step is to create these files using the Quartus II software.

2.5 Discussion

Researching on this topic is totally worth it. Partial reconfiguration can be used in time critical systems, like communication systems, where an abrupt stop of a data flow is critical. It is a nice way to improve configuration speed, because another part can still run. The field of partial reconfiguration is still a new topic in research and development and yet it is not integrated in industrial products. The main disadvantages are the cost of a special FPGA and special software instead of using an cheaper fully developed ASIC. Also a lot more time must be spent on validation before integrating such FPGAs into industrial products.

3 Fault simulation

3.1 Prerequisites

The hardware used in this part of the project was a former development kit from Altera DE2-115 with an 60-nm Cyclone IV E FPGA.

- 115.000 logic elements
- 4 Mbits embedded memory
- 60-nm low power process
- low energy consumption

In addition to the software mentioned in section 1, there is some special software required for this part of the project:

- C++ compiler e.g. MinGW to compile C++ files to *.exe files
- USB-to-Serial driver to get a connection between dev kit to pc via serial interface RS232

Also a couple of files from a former project of two students were provided. This part of the project is fully based on the former work of two students.

3.2 Idea and benefit

The idea of fault simulation is to detect stuck-at faults in an efficient way in hardware. This increases the fault detection speed in contrast to software. The concept is to compare a test pattern on a original circuit under test with a changed circuit. Therefore a special test pattern is loaded to the RAM of the FPGA via USB Blaster. One circuit without fault pattern and one with changed logical cells is generated. The output of both circuits is lead to a logical XOR gate which identifies if an error is detected or not. If the output of both circuits distinguishes the XOR gate give a logical one, hence a fault. Otherwise there was no fault or there is no fault detected. This output is saved into a FIFO (first in first out) memory and then transmitted to the pc for documentation. For this communication link the USB-to-Serial connection is needed.
The idea of the Maximum fanout free cones (MFFC), adopted from the former work, is to assign as many as possible gates to one logic cell. This would reduce the size used for this circuits. The creation of maximum fanout free cones is an algorithm to minimize the number of gates in one logic cell. The goal and benefit of this feature is a very fast and efficient way to detect faults in hardware.

3.3 Preparation and process to execute the fault simulation

After setting up the software prerequisites you can try to run the project. This list shows the process to execute the fault simulation on the FPGA:

1. compile quartus project
   - open 'FaultInjectionTester.qar' with Quartus II
   - compile this project
   - close Quartus II
2. create IRA.exe in root folder (batch file)
   - open command window and change directory to FaultConfiguration
   - execute: *"g++ IRA.cpp CGate.cpp CGate.h CComponent.h -o ../IRA.exe"*
3. create FaultInjectionTester.exe in root folder (batch file)
   - open command window and change directory to FPGACommunication
   - execute: *"g++ FaultInjectionTester.cpp PcFPGACommunication.cpp PcFPGACommunication.h -o ../FaultInjectionTester.exe"*
4. connect board and pc via USB Blaster
5. connect board and pc via RS232
6. execute IRA.exe [path to quartus], e.g. IRA.exe "C://Program Files//Altera"

If it all worked well, the pc programs and tests the FPGA. The communication link is via the RS232 interface, for which the usb to serial driver is necessary.
Otherwise following common errors can occur while executing:

- "Can not find quartus_cdb"
- "Inconsistent set or reset compile started variable"

For a solution see 3.5.

3.4 Implementation and improvement

To enhance the existing project a batch file was created which automates the process of generating the executable files. This makes it easier in a more convenient way to set up the project and make it ready for execution. Another step in the working progress was to change absolute paths to relative paths, so it can be used by everyone without checking the cpp program files. Then an error was detected in the design flow which leads to false results. This error cost very much time to find and it must be figured on finding other errors in the design flow. This shows that the previous work was not final and that there could be more errors.
To keep track of the execution and of the errors a logfile is created while execution. This

logfile makes it easier to track the progress and find errors in an easy way. A bunch of weeks was invested on debugging and struggling with several software installation problems to run the projects files on the FPGA.
The improvements and changes which were made in an overview:

- changed absolute paths to relative paths
- code debugging
- clear faults in design flow
- created logfile to keep track of the progress
- create a batch file to automate process

3.5 Problems and troubleshooting

Debugging and troubleshooting several errors became the main part of this project. Here are the problems mentioned which occurred during the project.

1. driver installation of USB Blaster II
2. driver installation of FTDI Chip for USB-Serial converter
3. error in the design flow
4. errors while execution
   a) "inconsistant set or reset of compile started variable"
   b) "quartus_pgm could not be found" - [insert SCREENSHOT]

1. Also it seems to be intuitive to install a USB driver, it was quite difficult to install in Windows 8.1 x64 pro. Although it is an official driver from Altera Windows 8.1 x64 pro does not support this driver because it has no driver signature. First of all the following problem was encountered : Neither the installation of the driver on the system cd nor the newest driver from alteras homepage could solve this problem. The attempt of windows to install this driver ended in the error in fig. 2.

After searching for a while in alteras forum and the internet to this problem, the solution was found on: ***http://altera-guide.blogspot.de/2012/11/many-collage-students-find-ourselves.htmlcomment-form***
Figure 8 shows the screenshot from the blog entry to solve this problem.

After successfully installing the driver, another problem may occur while starting the driver, see fig.3:

2. If the USB Blaster driver was successfully installed, there came another problem concerning the USB-to-Serial conversion for the communication link between pc and dev kit. The driver could not be installed and hence it could not be started see Fig. 4 and 5.

Figure 1: failed installation of usb blaster driver

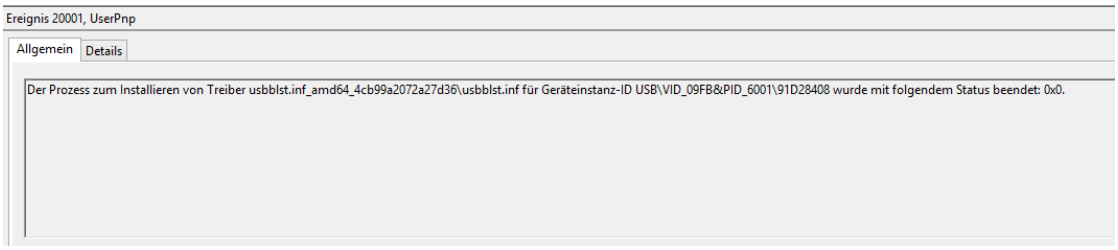

Figure 2: AlteraUSBBlaster install not successfull

If you try to install the driver with the standard windows application you run into the problem seen in Fig. 6.

In the end the driver for the FTDI Chip was searched in the internet and installed.

3. There was an error in the design flow of the former work. It was realised that there was no compilation after changing LUTs of circuit under test.

If all the troubleshooting about the installation of different drivers were made another error was detected while executing the project.

4a The error occurring while execution was: "inconsistant set or reset of compile started variable" see Fig. 7.

To solve the problem of "inconsistant set" you have to follow this:

1. clear out the "db" and "incremental_db" folder in "FaultInjectionTester_restored" folder

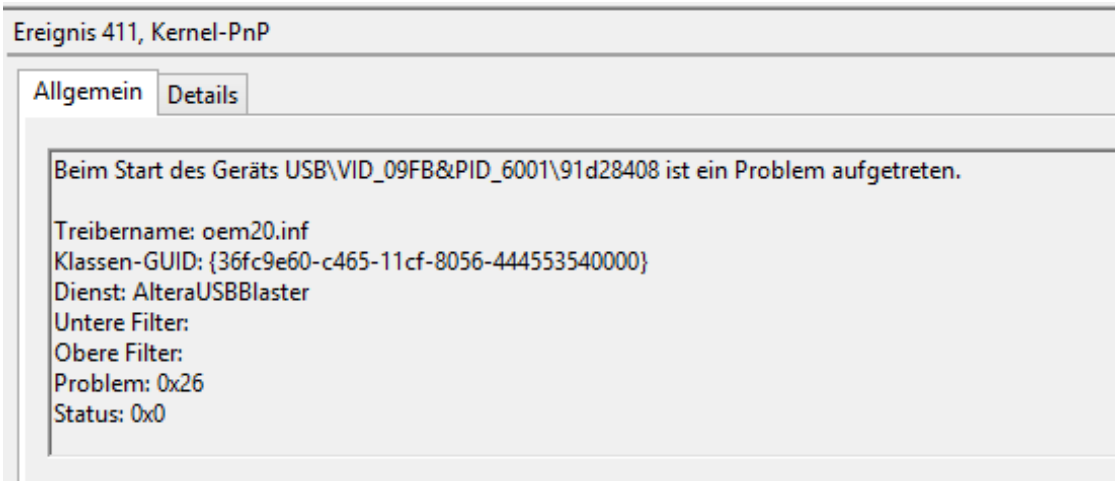

Figure 3: AlteraUSBBlaster problem

2. recompile Quartus project

3. execute IRA.exe once more

4b. If an error message appears which states that a "quartus_pgm" or "quartus_cdb could not be found", you have to check the path of your quartus installation.

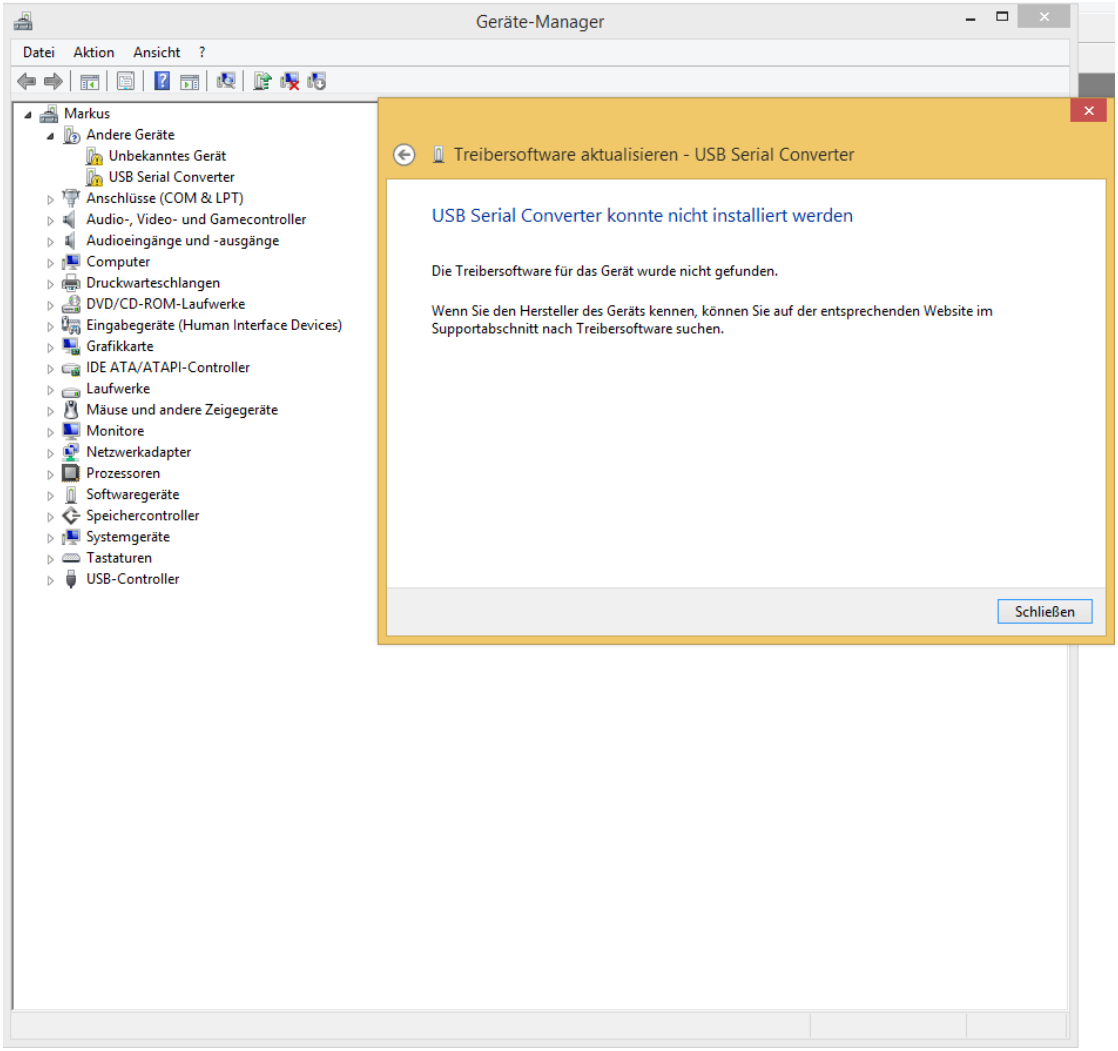

Figure 4: USB Serial Converterter could not be installed

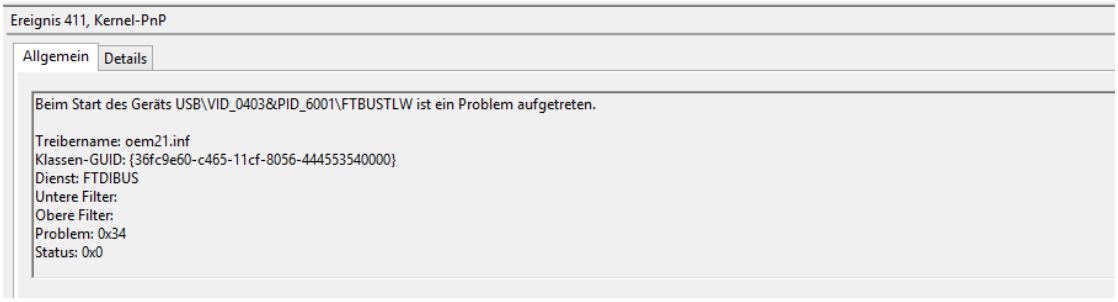

Figure 5: FTDI chip problem

Figure 6: Driver install not successful

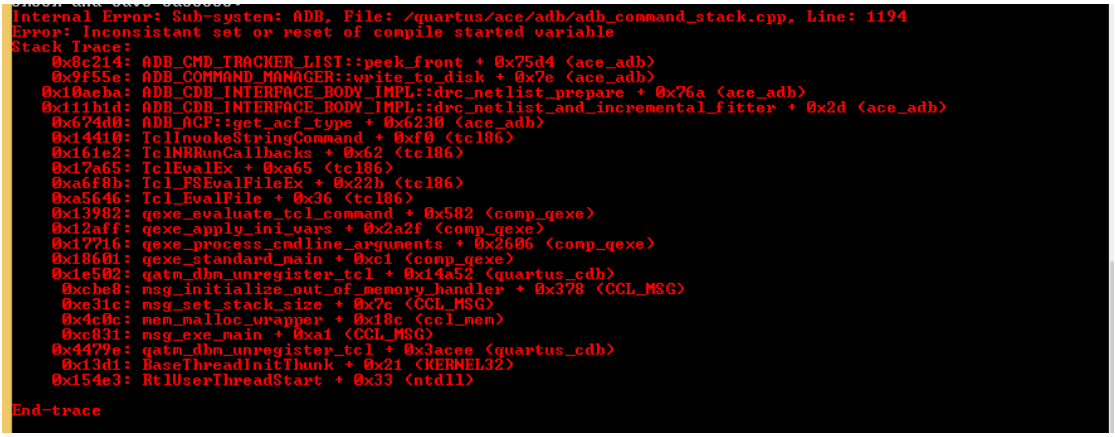

Figure 7: inconsistent set

3.6 Discussion and future work

Debugging and troubleshooting become the main part of the project. A lot of problems occurred during this work. These problems prevented a learning process in vhdl, which was the intend to do a project at the chair of computer architecture. The second part of the project was a project in the programming language of C++ and hence none of hardware programming.
The workstation for this project was a desktop pc and not a laptop, which makes it difficult to share the problems like driver issues with the other guys.
The best way to enhance this project were to set up the whole project on a virtual machine. So there could not occur any software issues and the project can be run independently of any operating system. There should be only one supported software version of Quartus II and QSYS and also a version control feature for the project. The version control is needed if a group should work with this project. This setup would have taken too much time in the workload of this project but it would be a nice part for future work.

Installation Guide for Windows 8, and 8.1 (Windows 10 should be similar)

Many college students find ourselves using the Altera DE2 board in our circuit design classes.
Installing Quartus is an easy and pretty simple task; however, installing the USB Blaster drivers for the DE2 is a huge pain.

Recently I moved to Windows 8 and was presented with the task of re-installing all of my programs and with that trying to get the USB Blaster to work. Doing so on Windows 7 is quite simple but with the new Driver Signature Enforcement in Windows 8 the drivers that Altera provides refuse to install properly.
As a result I searched around and found a solution on Altera's forum:

http://alteraforums.net/forum/showthread.php?p=157605

Looking at the final post in the thread by Blackwater we are presented with a quick and dirty solution that doesn't require you to install any additional software.

Here are the steps:

1. Windows Key + R.
2. Enter shutdown.exe /r /o /f /t 00
3. Click the "OK" button
4. System will restart to a "Choose an option" screen
5. Select "Troubleshoot" from "Choose an option" screen
6. Select "Advanced options" from "Troubleshoot" screen
7. Select "Windows Startup Settings" from "Advanced options" screen
8. Click "Restart" button
9. System will restart to "Advanced Boot Options" screen
10. Select "Disable Driver Signature Enforcement"
11. Once the system starts, install the "USB Blaster" driver as you would on Windows 7

Here is a link to the 32bit and 64bit drivers

https://www.dropbox.com/sh/oh8siwfct6c4ruy/6gg83GH5Ax

To install these:

1. Open device manager
2. Find USB Blaster
3. Right Click and select "Update Driver Software"
4. In the popup window select "Browse my computer for driver software"
5. Browse and find the location of the files that you downloaded earlier selecting "x64" or "x86" depending on your system.
6. Go next and install. If you get a popup saying that the driver is unsigned just ignore it and click continue.
7. Hopefully it works out and now you are ready to use your DE2 with your Windows 8 system

Figure 8: steps to install the usb driver