# Partial reconfiguration and fault simulation with Altera Cyclone V FPGA

**Students project at the chair of computer architecture university freiburg**

Markus Weiß

February 24, 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 needed to generate VHDL- or Verilog programs which can be used by a fpga. The 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 of different components. The developed VHDL code from Quartus II software can be imported to QSYS to connect different components, e.g. on the DE-SoC1. For an example QSYS has to initiate, configure and connect the HPS to the FPGA if they need to communicate.
USB driver for USB Blaster is needed to program the fpga with the computer.

# 2 Partial reconfiguration

## 2.1 Prerequisites for the project of partial reconfiguration

The **hardware** used for this project was a development kit board from altera, namely "Terasic DE1-SoC Development Kit". This board consist of the following core components: a FPGA, named Altera Cyclone V FPGA (5SCEMA5F31C6N), a dual-core Cortex-A9 (HPS), 1GB DDR3 and 64MB SDRAM and different interfaces (USB,JTAG,i2c). The communication interfaces are a on board USB-Blaster II, USB 2.0 ports, UART interface, Ethernet port and extended module for gpio pins.
The fpga is the mainly used component of the development kit board. Here are some useful facts about the fpga Cyclone V:

- 85000 logic cells on the board
- 4450 kb embedded memory
- access to an external 64 MB static random access memory (SRAM)
- low energy consumption
- standard clock frequency is 50 MHz

The Cyclone V is configured with a JTAG interface, where a bit stream is loaded to the board. This bit stream is lost once the energy is off. Also available is the option to program an in-system NOR flash memory, which safes the bit stream while power is taken off.
Another way to program the fpga is to use the HPS. This is done while the boot process with "U-Boot" or the "preloader", but it also works if the operating system is fully booted.
The **preloader** initiates the clock, multiplexing of pins, configuration of the main memory and load the U-Boot. The configuration of the main memory of the HPS and the

AXI Bridges is important, 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.
The **Avalon-Memory-Mapped-Interface** 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. The master put a request to the slave, which read or write special addresses of the memory. This interface is required to get access from the FPGA to the main memory of the HPS.
To work with an external C/C++ program an operation system for the hps is needed. The chosen OS is a linux system.

## 2.2 The idea and the benefit of partial reconfiguration

The idea of partial reconfiguration is highly desirable for some special purposes. With this feature it is easier to change designs or improve the the functionality without stopping its working progress. Some applications need to change some logic while another part is not allowed to stop, e.g. the flow of data. Therefore this function is desirable in the field of communication systems and high-performance computing platforms. A benefit of this idea is that the number of devices can be scaled down, therefore the power consumption and of course the cost. The tasks another fpga has handled before can now be handled by the same fpga, only another part of the logic cells take this.
Thanks to this feature the fpga can handle a new input while another part still executes its commands.
This reconfiguration is initialized by an external processor on the development kit (hps) and programs the desired locig cells of the fpga while the other part remains in its process. The initiation is done by an internal host which runs a C/C++ program. This C/C++ program is stored in the RAM (read access memory) of the development kit board.
Partial reconfiguration could be combined with the fault simulation on fpga for a efficient way to detect logic errors.

## 2.3 Design flow and implementation of partial reconfiguration

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.
There have to be two different regions in the top level design, one static and one region for the partial reconfiguration. Two led pattern are loaded to this regions. 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 or partial reconfiguration region which let us check if a part of the fpga is reconfigured.
A wrapper region is generated to ensure that all of the PR personas have the same connection to the static region. This is done by creating dummy ports.
For partial reconfiguration all non-global inputs of the pr regions must be freezed. Freez-

ing 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 created.
The projects idea was a partial reconfiguration with an internal host, which initializes the reconfiguration with a c/c++ program. To do so, 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 crucial point. The design flow used in this project was as follows:

1. planning the system for partial reconfiguration
2. identify which blocks to be partially reconfigured → led pattern for the led on DE1-SoC board
3. code the design in Quartus II
4. develop the personas
5. generate static and pr region
6. assign partitions to logiclock regions
7. create necessary files to program fpga
8. program fpga

The FPGA development process is done by Quartus II and QSYS. Quartus II creates the FPGA components 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. The next step of booting is the "universal boot loader" which configures the fpga and load the "linux-kernel". The Avalon-MM-Interface is required to grant the fpga access to the main memory of the HPS.

## 2.4 Problems during the first project of partial reconfiguration

**Partial reconfiguration is not supported by the Cyclone V fpga**. Following problems occurred:

- no permission to use partial reconfiguration feature in Quartus Software; so a new license was inquired.
- the fpga Cyclone V does not support the feature of partial reconfiguration; this hardware problem could not be solved because a new special fpga was not affordable for the chair

After working 8-10 weeks in the field of partial reconfiguration the project has to be stopped. At this time the status quo was:

- literature survey
- soft- and hardware prerequisites
- project design flow
- creating different revisions for partial reconfiguration in Quartus II
- VHDL design
  - top level
  - wrapper
  - freeze region
  - static region + personas
  - partial reconfiguration region + personas
- pin assignments
- file creation for programming the fpga (.pmsf, .msf, .sof, .rbf)

# 3 Fault simulation

## 3.1 Prerequisites for the project of fault simulation

**Hardware** used in this project was a former dev kit from altera with a fpga Cyclone IV. In addition to the former mentioned software there is some **special Software** required for this project:

- C++ compiler e.g. MinGW → compile C++ file to *.exe file
- USB-to-Serial driver → get a connection from dev kit to pc via RS232

Also a couple of files from a former work were provided. The former work was also a project by two students at the chair of computer architecture.

## 3.2 Idea of fault simulations in hardware

The idea of fault simulation is to detect stuck-at faults with the fpga. A special test pattern is loaded to the ram of the fpga. This is done by the communication from pc to the fpga via usb blaster. The circuit under test is realized by one circuit without fault pattern and one with changed lcells. The changed logic cells are loaded with test pattern to compare both. The output of both circuits is lead to a logical XOR gate. If the outputs of both circuits distinguishes the XOR gate give a logical one, hence a fault. Otherwise no fault is detected. This output is saved in a fifo and then transmitted to the pc for documentation purposes via the serial interface RS232. The idea of the Maximum fanout free cones (MFFC), adopted from a former project, is to assign as many as possible gates to one logic cell. The creation of maximum fanout free cones is an algorithm to

minimize the number of gates in one logic cell. This algorithm is implemented in quartus by a former work. The original circuit under test is compared to a modified circuit. The look-up table (LUT) can be calculated for every locig cell in quartus. The idea of the fault injection test is an efficient way to detect faults in hardware. The evaluation is done over the rs232 interface and a personal computer. The goal is a very fast way to detect those faults.

### 3.3 preparation to execute the fault simulation on the development kit

After setting up the software prerequisites you have to change the absolute paths inside the C++ files to the desired programs. This list shows the process to execute the fault simulation on the fpga:

1. open 'FaultInjectionTester.qar' with Quartus II
2. compile this project
3. close Quartus II
4. open command window and change directory to FaultConfiguration
5. execute: *g++ IRA.cpp CGate.cpp CGate.h CComponent.h -o ..IRA.exe*
6. open command window and change directory to FPGACommunication
7. execute: *g++ FaultInjectionTester.cpp PcFPGACommunication.cpp PcFPGACommunication.h -o .."FaultInjectionTester.exe"*
8. Connect board and pc via USB Blaster to programm
9. connect board and pc via RS232 to communicate

### 3.4 process

1. compile quartus project
2. create IRA.exe in root folder (make.batch file)
3. create FaultInjectionTester.exe in root folder (make.batch file)
4. execute IRA.exe [path to quartus], e.g. IRA.exe "C://Program Files//Altera"), beware of whitespaces then you have to write double frontslash, otherwise normal frontslash would be enough.

If it all work well, the pc programms the fpga and tests it over the RS232 communication interface.
explain the complete process:

1. TCL script

2. batch
3. quartus project
4. c++ files
5. exe files
6. vhdl files

Otherwise following common erros occur:

- Inconsistent set or reset compile started variable - clear out db and incrementaldb folder in FaultInjectionTesterrestored folder and recompile quartus project
- Can not find quartuscdb - if quartus is not installed on C: you have to change the HDD constant in FPGACommunication/PcFPGACommunication.cpp and FaultConfiguration/IRA.cpp to the drive where your quartus is installed.

## 3.5 Implementation

approach, vorgehensweise -> adaption of the previous work. was man alles beachten muss, was man alles aendern muss... pfade aendern, treiber installieren (-> nicht trivial !!), tcl, batch, c programm, vhdl, qsys, quartus programm verstehen, was was macht. design flow, work flow, compilation. what to do... what changed...

## 3.6 Problems

Here are the problems which occur during the project.

1. driver installation of USB blaster II and FTDI Chip for USB-Serial converter
2. error in design flow
3. error while execution: inconsistent set and quartus could not be fond

Also it seems to be intuitive to install a USB driver, it was quite difficult to install under Win 8.1 because the delivered driver has no driver signature which is required for Win 8.1. To install a driver without signature you have to follow these steps to install it anyway. First of all i encountered the problem of driver installation. Although it seems to be trivial, if you encounter this problem: you have to follow these steps to install the usb driver:

Although it is an official driver from altera i was struggling with this problem a few days to find out, that windows 8.1 does not support this official driver. The solution is found on: ***http://altera-guide.blogspot.de/2012/11/many-collage-students-find-ourselves.htmlcomme*** ***form*** The solution for the inconsistent set error would be to clear out 'db' and 'incremental db' in 'FaultInjectiontester restored' folder and recompile the quartus project.


Figure 1: failed installation of usb blaster driver

If a 'quartus' execution file could not be found, you have to check the paths entered in FPGACommunication.cpp and IRA.cpp and change them. A lot of problems occurred

during the work on the project. These problems prevented that vhdl could be learned, which was the intend to do a project at the chair of computer architecture. The second project was nearly only a project in the programming language of C++ and no more hardware programming. The workstation for this project was my desktop pc and not my laptop. So this makes it difficult to share the problem among the other guys. The best way to enhance this project were to redo the whole project. The idea was to set up a whole new project which runs fine on my machine but then it would run fine on my work station but not in common. So there would be a good improvement, to set up a virtual machine with software which runs independent of the users operating system. There would be only one supported software version of Quartus II and QSYS, the driver could be set up once. This setup would have taken too much time in the workload of this project. This took a very long time, because it is so complicated and not documented and not trivial. Also there was a huge error in the design flow, which took very long to detect. Than there was the problems discussed earlier. Debugging and troubleshooting become the main part of the project.
The improvements and changes which were made:

- changed absolute paths to relative paths

- code debugging
- error in design flow detected: re-compilation after changing the LUTs is missing
- create a batch file to automate process

Further improvements for future work is to setup the program on a virtual machine to become independent of operating system and thus driver installation. Also a version control for this program would be great for future works and a nice structure of the project. In this project is no structure and it makes it very difficult to understand whats going on.

# 4 Appendix

## 4.1 A - Problems and Errors

# 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 2: steps to install the usb driver

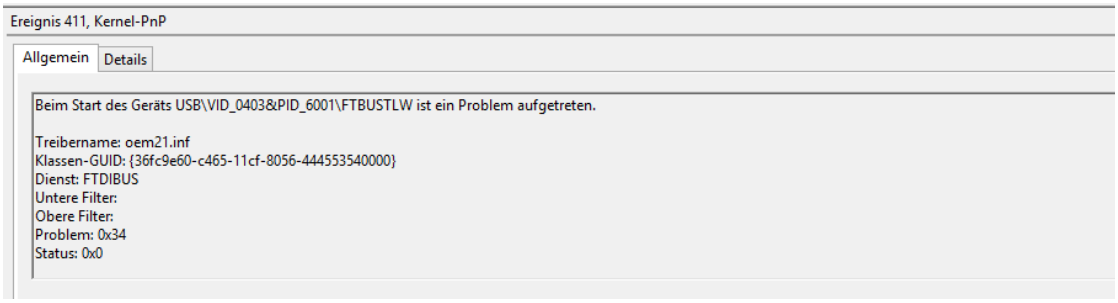


Figure 3: FTDI chip problem


Figure 4: Driver install not successful

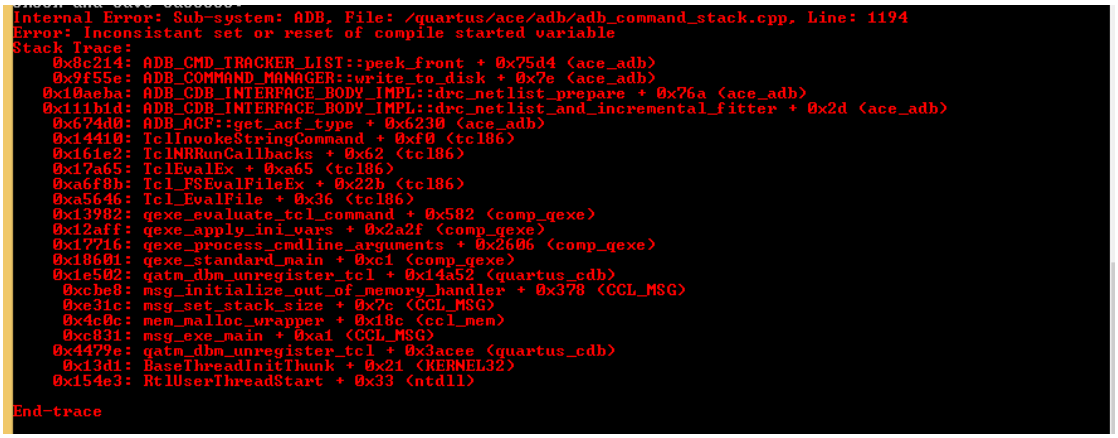


Figure 5: inconsistent set

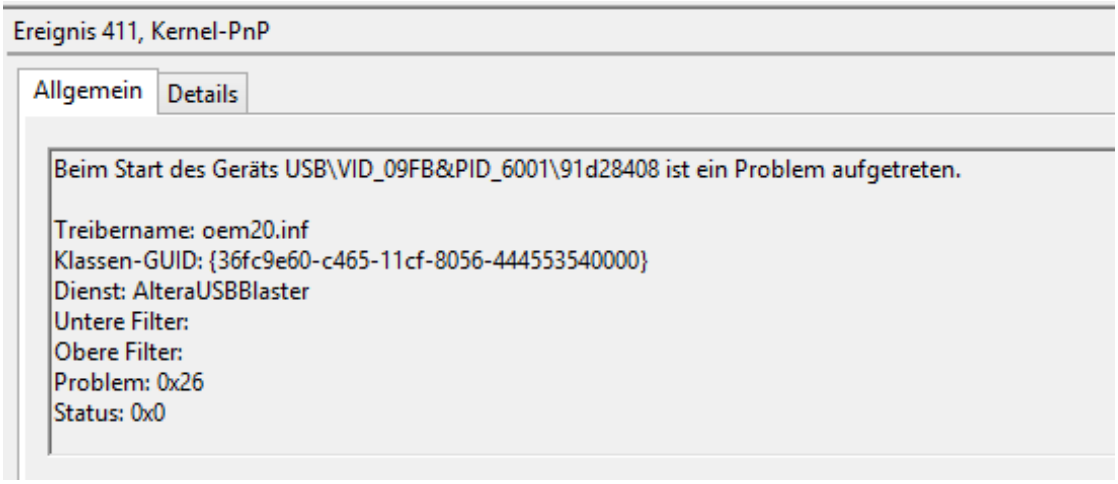


Figure 6: AlteraUSBBlaster problem

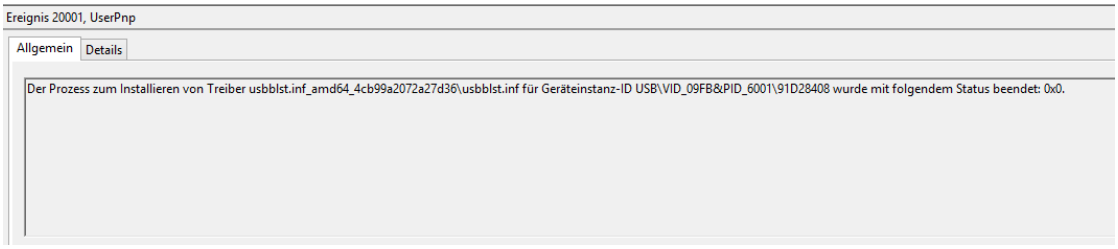


Figure 7: AlteraUSBBlaster install not successfull

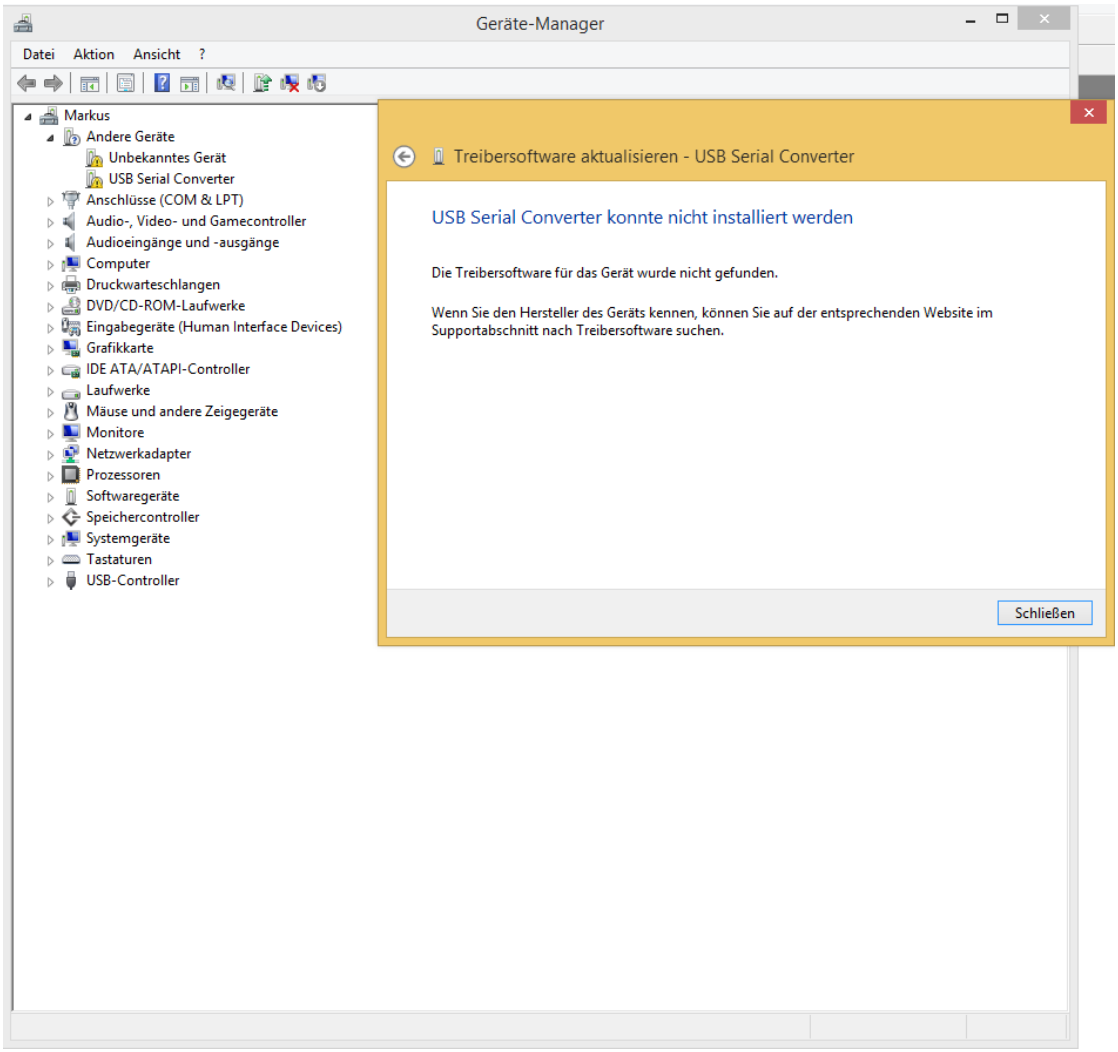


Figure 8: USB Serial Converterter could not be installed