Partial reconfiguration and fault simulation with Altera Cyclone fpga

Students project at the chair of computer architecture university freiburg

Markus Weiß

February 25, 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 was needed to generate VHDL- or Verilog files which were used by a fpga (field programmable gate array). 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 if they need to 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 from altera needs to be installed. The installation of the Altera USB Blaster driver was not so intuitive as expected (see problems).
The following revisions and version of the 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 (partial reconfiguration)
- usb-to-serial converter driver (fault simulation)

2 Partial reconfiguration

2.1 Prerequisites

The hardware used in this 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 a 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 of the development kit board and therefore here are some useful facts about it (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

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, which safes the bit stream while power is taken off, can be loaded with this bit stream.
Another way to program 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.
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.
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 start a C++ program on the board an operation system for the hps is needed. The used operating system was a small sized portable linux system wihch was loaded to an sd card and started from there.

2.2 The 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 a fpga without interrupting its work progress. Special applications need to change some logic on a fpga while another part of the fpga is not allowed to stop. For example the flow of data should not be stopped while another part is changing something. 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 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 it.
Thanks to this feature the fpga can handle a new input while another part still executes its commands.
This reconfiguration is either initialized by an external host or an internal one. With the internal host, a processor on the development kit (hps) initializes the process and starts to program the desired locig cells of the fpga while the other part remains in its process. This 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 an efficient way to detect logic errors.

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 the initialized process and one region, which can be reconfigurated on the fly. 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 or partial reconfiguration region which let us check if a part of the fpga is reconfigured.
To avoid in- and output errors while reconfiguration, 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 is 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.
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:

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

There occurred two crucial problems 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 special 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 cyclone V fpga. 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)

Researching on this topic of partial reconfiguration it worth it. Partial reconfiguration can be used in time critical systems, like communication systems, where an abrupt stop of a data flow is critical. The field of partial reconfiguration is still a new topic for researcher and yet it is not integrated in industrial products. The main disadvantage is the cost of a special fpga instead of using an cheaper ASIC and a lot more time must be spent on validation.

3 Fault simulation

3.1 Prerequisites

The hardware used in this part of the project was a former dev kit from altera with a fpga Cyclone IV.
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 → compile C++ file to *.exe file
- USB-to-Serial driver → get a connection for communication purpose from dev kit to pc via RS232

Also a couple of files from a former project of two students were provided.

3.2 Idea and benefit

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 and process to execute the fault simulation

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. compile quartus project
    - open 'FaultInjectionTester.qar' with Quartus II
    - compile this project
    - close Quartus II

2. create IRA.exe in root folder (make.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 (make.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:

- "Inconsistent set or reset compile started variable" → clear out "db" and "incremental_db" folder in "FaultInjectionTester_restored" folder and recompile quartus project
- "Can not find quartus_cdb" → you have to change the HDD constant in FPGACommunication/PcFPGACommunication.cpp and FaultConfiguration/IRA.cpp to the drive where your quartus is installed.

3.4 Implementation

To enhance the existing project a batch file was created which automates the process of generating the executable file from the cpp 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 errors 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 fully completed and debugged. It takes a lot of time to debug the former code. A bunch of weeks was invested on debugging and struggling with several software installation problems to execute the project. This leads to the idea of redoing the whole project and setting it up to an independent machine like a virtual machine with a version control of the project. This would ensure that everybody can use it without special requirements. A version control would be a good start to get a nice structure of the project workspace.

3.5 Problems and its solution

Debugging and to solve problems on installation and design flows become the main part of the project work. 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

Figure 1: failed installation of usb blaster driver

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.htmlcomment-form***

A lot of problems occurred during this work. 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

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

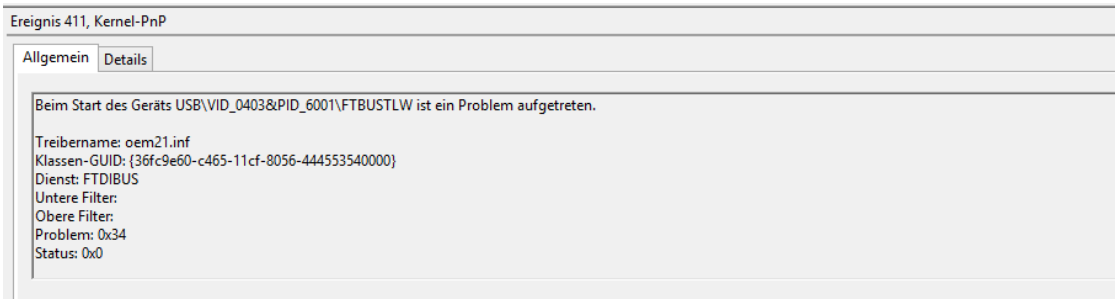

Figure 3: FTDI chip problem

Figure 4: Driver install not successful

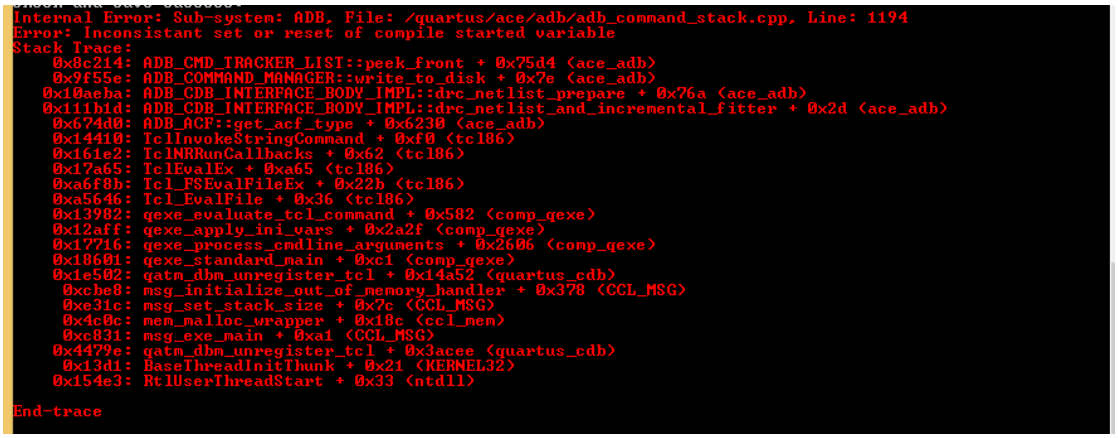

Figure 5: inconsistent set

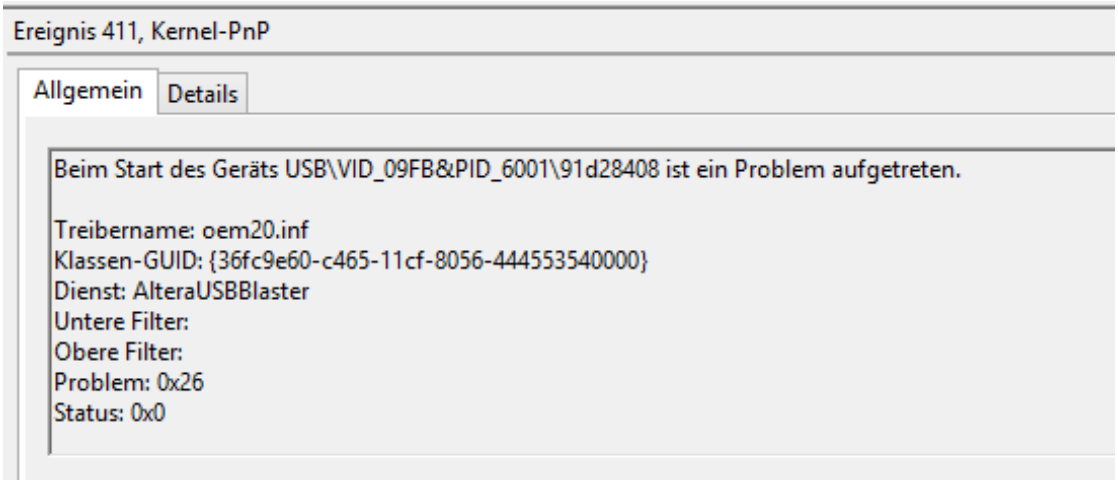

Figure 6: AlteraUSBBlaster problem

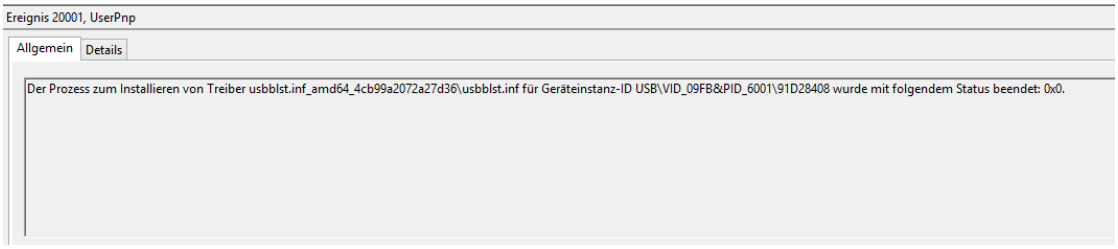

Figure 7: AlteraUSBBlaster install not successfull

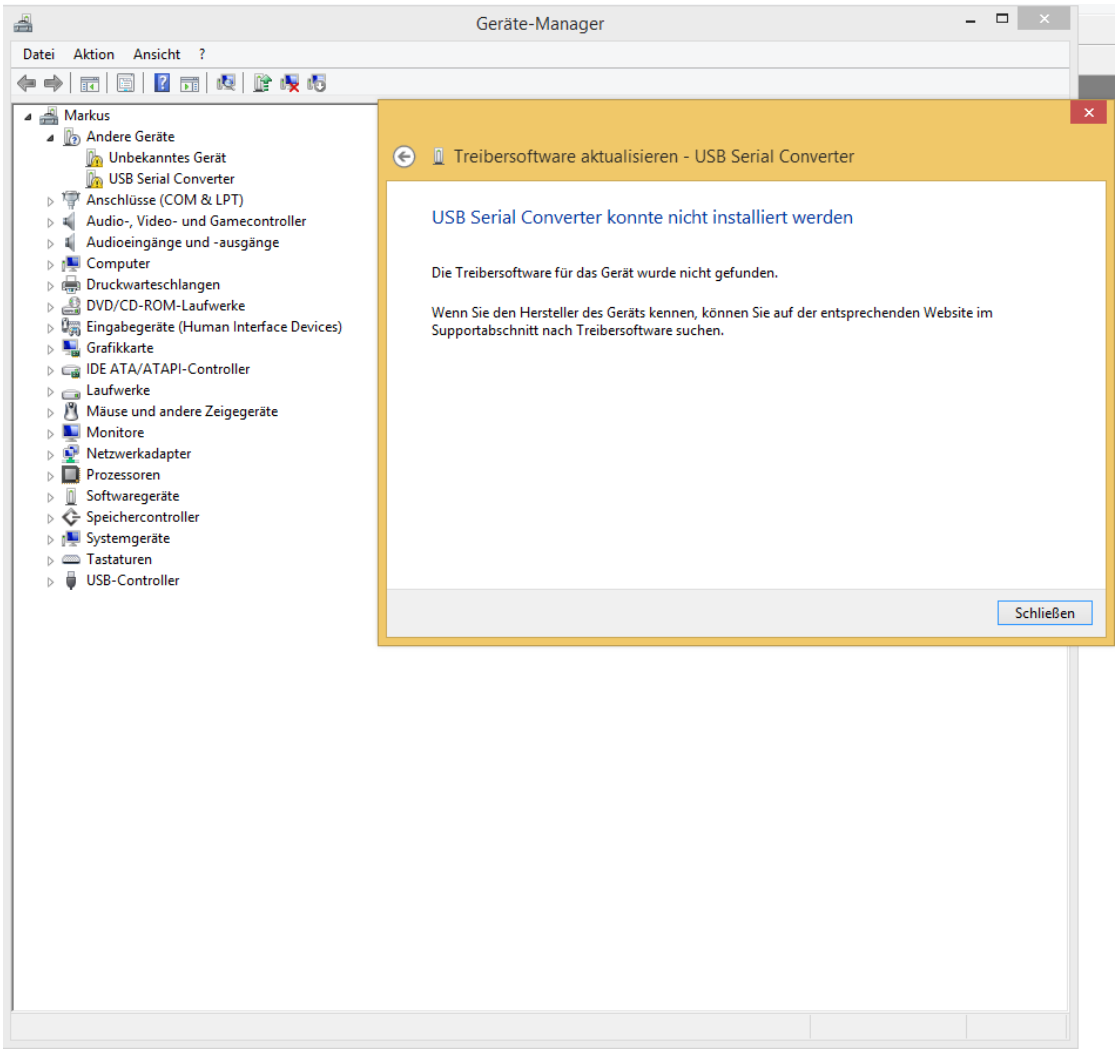

Figure 8: USB Serial Converterter could not be installed