Partial reconfiguration and fault simulation with Altera Cyclone V FPGA

Students project at the chair of computer architecture university freiburg

Markus Weiß

February 20, 2016

Contents

1 HW/SW prerequisite for this project

The Hard- and Software prerequisites are the same for both projects, so here is a short summary of the used hard- and software.

1.1 Hardware

The hardware used in this project were a "Terasic DE1-SoC Development Kit". This development kit consists of the following core components: a FPGA, named Altera Cyclone V FPGA (5SCEMA5F31C6N), a dual-core Cortex A9 processor, also named Hard-Processor-System (HPS), different memories and interfaces (USB,JTAG,i2c). The communication interfaces are a on board USB-Blaster II, USB 2.0 ports, UART, Ethernet port and extended module for gpio pins.
The fpga is the heart of the development kit and the mainly used component of the development kit board. Some 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.

1.2 Software

The software used in this project 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, therefore cost of developer 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. QSYS has to initiate, configure and connect the HPS to the FPGA if they have to communication between each other.

2 Partial reconfiguration

2.1 The idea and the benefit of partial reconfiguration

The idea to re-configurate a fpga while the rest of it is still running is highly desirable. 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 partial reconfiguration the fpga can handle a new input while another part still executes its commands. This reconfiguration is initialised 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 intiation is done by an internal host which runs a C/C++ programm. This C/C++ programm is stored in the RAM (read access memory) of the development kit board.
Partial reconfiguration should be combined with the fault simulation on fpga. therefore a very efficient way to detect logic errors with the fpga is established.

2.2 Approach to integrate PR - Implementation

The approach for the partial reconfiguration was, to create two led pattern to check if the reconfiguration is done properly. One for the static and one for the dynamic region. therefore several files are nedded.

1. create a simple led pattern for static region
2. second wrapper under freeze region
3. one more led pattern for dynamic region

4. determine LogicLock regions -> what is that?

5. check warnings

6. connect I/O signals with pins (QSYS)

The idea of the project was to implement an internal host, which initializes the reconfiguration with a c/c++ program. To do so, the processor has to communicate with the fpga. **where is the internal host? how dows it work? who communicate with whom?**
The communication between the hard processor system and the fpga, the main memory and in- and output interfaces is an important point. **communication link between processor, fpga, interfaces...** There are different options for the communication between HPS and FPGA.

1. Advanced extensible interface bus bridges (AXI)
   a) Lightweight HPS-to-FPGA bridge
   b) HPS-to-FPGA bridge
   c) FPGA-to-HPS bridge
2. FPGA-to-SDRAM

FPGA access the memory and load the bitstream safed in the sdram?? **which option is used in our project? which one should be used?**

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.
Project design flow, hierarchy etc, code citation. Which design blocks are necessary to program the fpga for partial reconfiguration. Wrapper, logic, etc. ...comment on created code.

2.3 Problems

After working a couple of weeks on the field of partial reconfiguration the project has to be stopped. Following problems occured:

- 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

The status quo at the time of cancellation is:

- literature survey
- top level design
- static region → ticker (still runing while reconfiguration)
- pr region → two different LED pattern; persona A, persona B
- wrapper design

2.4 Discussion

Partial reconfiguration can be used in time critical systems, like communication systems where a abrupt stop of a data flow is critical. The field of partial reconfiguration is still a new one to researcher. It is not integrated in industrial products yet, because if a special fpga is needed instead of an ASIC it is not very cost efficient.

3 Fault simulation

3.1 Idea of fault simulation

The idea of fault simulation is to detect so called stuck-at faults with the fpga. Its purpose is, if a faulty pattern can be recognised or not. A special pattern is loaded to the ram of the fpga and processed by the hps. 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 with test pattern to compare both. The output of both circuits is lead to a logical XOR gatter. If the output distinguishes each other the XOR gatter give a logical one, hence a fault. This output is safed in a fifo and then transmitted to the pc for documentation purposes via the serial interface RS232.**diagram...** Efficient fault simulation in hardware FPGA. modification of the circuit with special changes to the look up tables. evaluation and pc communication with serial interface rs232.
lcell

- smallest logic cell
- maximal four inputs
- gatter are ordered in a logic cell
- every lcell have a look-up table, which defines the output
- the function of a particular lcell can be changed by hand

The idea of the Maximum fanout free cones (MFFC) is to assign as many as possible gatter to one logic cell. The creattion of maximum fanout free cones is an algorithm to minimize the number of gatter in one logic cell. this algorithm is implemented in quartus by a proper work. the original circuit under test is compared to a modified circuit. The 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 a fpga. the evaluation is done over the rs232 interface and a personal computer. the goal is a very fast way to detect those faults.

3.2 Special software requirements

In addition to the mentioned software in the beginning some special software is required:

- C++ compiler e.g. MinGW
- USB driver for USB Blaster

Also it seems to be intuitive to install USB driver, it was quite difficult to install under Win 8.1 becuase 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.

1. ...

3.3 preparation to execute the fault simulation on the development kit

After installing a C++ compiler and the USB driver for the blaster, you have to change the absolute paths to the desired programs in the C++ files.

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

absolute path to installed software. troubleshooting for installation of usb blaster driver. debugging former code fractions. error in design flow (compilation missing after changing luts). creating batch file to automate it. relative paths to automate. further improvements: design for a virtual machine, version control.

3.7 Discussion

what can be changed, what is the point of work, what are the next steps. Which problems can be solved how? All in all i can say to work on a proper design it is very difficult. The communication link between RS232 and Baord work properly, but without the information on the Version of used software (quartus, qsys, operating system etc.) it is too difficult. Also the installation path should not be absolute. There were an error that the project was not compiled after changing the LUTs. Debugging and troubleshooting took too long to stay focused and motivated to this project.