TMAP: A Reconfigurability-Aware Technology Mapper

Karel Bruneel, Fatma Abouelella and Dirk Stroobandt

karel.bruneel@ugnet.be

HES-ELIS – Ghent University – Belgium

http://hes.elis.ugent.be/kbruneel

Abstract

We present an FPGA tool chain with a new technology mapper TMAP. The tool flow postpones parameter bounding until after place and route, enabling fast run-time specialization of FPGA configurations.

1. Introduction

The inherent reconfigurability of SRAM-based FPGAs enables the use of different configurations at different time intervals, each optimized for the specific task in the corresponding time interval. Optimized configurations are smaller and faster than their generic counterparts and require less power. Therefore, they use the FPGA's resources more efficiently. However, at the time interval boundaries, the problem at hand will change and valuable resources will need to be used to generate or select a new configuration and reconfigure the FPGA.

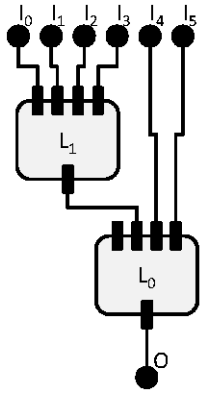
Conventional synthesis tools generate FPGA configurations from scratch. They use heuristics to solve NP-complete problems like placement and routing. Hence, generating a new configuration requires huge amounts of resources. This makes run-time generation of configurations with conventional tools inefficient for most applications.

In [1], we have shown that it is possible to generate an arbitrary new configuration with significantly less resources and without sacrificing much of the quality if subsequent data manipulations only differ in a set of parameter values. In between these parameter changes, the parameter set remains at constant values during relatively long time intervals. In this paper, we descibe a tool flow that targets commercial FPGAs. The tool flow automatically maps a high-level description of an application to a master configuration and a specialization procedure, that is responsible for updating the FPGA's configuration memory on a parameter change.

2. Tunable LUT mapping

The basis of our reconfiguration method is the notion that the bits that form a LUT truth table can be expressed as a Boolean function of a set of parameters, called *tuning functions*. A LUT of which the truth table bits are expressed as tuning functions is called a *tunable LUT*, or TLUT [1]. A circuit containing TLUTs is called a TLUT circuit. The most important property of such a TLUT circuit is that it can very rapidly be transformed into a regular LUT circuit for one specific set of parameter values by simply evaluating its tuning functions.

In [1] and [2], we presented generic methods for automatically generating a TLUT circuit from an arbitrary parameterizable circuit. Such a circuit contains *parameter inputs* which change values less often than the general inputs. The core of the method is TMAP, a reconfigurability-aware technology mapper. An important property of the TLUT circuits produced by TMAP is that they generally use less TLUTs than a generic FPGA implementation uses regular LUTs, while maintaining full flexibility of assigning values to the parameters.



L_1	L_0
0	0
S_0S_1	$S_0\overline{S}_1S_2$
$\overline{S}_0S_1$	$\overline{S}_0\overline{S}_1S_2$
S_1	$\overline{S}_1S_2$
$S_0\overline{S}_1$	$\overline{S}_2$
S_0	$S_0\overline{S}_1+\overline{S}_2$
$\overline{S}_0S_1+S_0\overline{S}_1$	$\overline{S}_0\overline{S}_1+\overline{S}_2$
S_0+S_1	$\overline{S}_1+\overline{S}_2$
$\overline{S}_0\overline{S}_1$	0
$S_0S_1+\overline{S}_0\overline{S}_1$	$S_0\overline{S}_1S_2$
$\overline{S}_0$	$\overline{S}_0\overline{S}_1S_2$
$\overline{S}_0+S_1$	$\overline{S}_1S_2$
$\overline{S}_1$	$\overline{S}_2$
$S_0+\overline{S}_1$	$S_0\overline{S}_1+\overline{S}_2$
$\overline{S}_0+\overline{S}_1$	$\overline{S}_0\overline{S}_1+\overline{S}_2$
1	$\overline{S}_1+\overline{S}_2$

Figure 1: TLUT circuit of the 6:1 multiplexer example and the corresponding tuning functions.

To illustrate the concept of TMAP, we will explain it on the example of a 6:1 multiplexer. The multiplexer has six data inputs (I_0 through I_5), three select inputs (S_0, S_1 and S_2) and one output (O). Without loss of generality, we choose the select inputs S_0, S_1 and S_2 as parameter inputs. One can easily see that a conventional technology mapper will need at least four 4-input LUTs, with fixed LUT truth tables, to implement this multiplexer. This is the generic FPGA implementation. In the case of a TLUT circuit, the parameter inputs can be included in the LUT function and do not have to be synthesized as real inputs. In this case, TMAP can implement the multiplexer using only two TLUTs, as shown in Fig. 1, a significant reduction of 50% in area for this example. Because we assume the target FPGA fabric has 4-input LUTs, there are 16 tuning functions associated to each TLUT's truth table, which express the functionality of the TLUT depending on the parameter inputs.

3. The TMAP tool flow

The tool flow used to map a parameterizable HDL design to a self-reconfiguring platform is shown in Fig. 2. The

self-reconfiguring platform contains two subsystems: the FPGA fabric and the *configuration manager*. Upon a change of a parameter value at run-time, the configuration manager will evaluate the tuning functions in order to obtain new truth tables and will then reconfigure the TLUTs. The tool flow produces both a master configuration, which is used to generate an initial bitstream to configure the FPGA at start-up, and a set of reconfiguration procedures, that serve as the basis for the configuration manager.

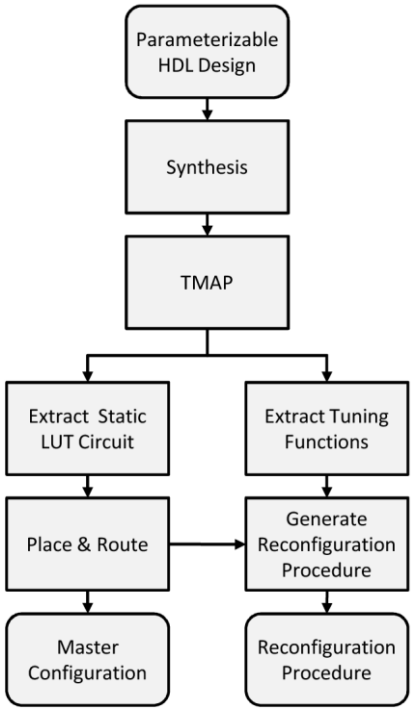


Figure 2: Tool flow for mapping a parameterizable HDL design to a self-reconfiguring platform.

After a conventional synthesis step, TMAP maps the design into a TLUT circuit. Because the parameters only affect the truth tables of the TLUTs we can extract a static LUT circuit from the TLUT circuit. Obtaining the static LUT circuit can simply be done by ignoring that the TLUT truth tables depend on the parameter values. The static LUT circuit can then be implemented off-line using conventional placement and routing tools. The output of this implementation process is the master configuration.

In order to change one specific TLUT's function according to new parameter values, we need to know the way its truth table is related to the parameter values, i.e., the tuning functions, and the location of the physical LUT that implements the TLUT under consideration. The combination of this information for all TLUTs is used to generate a C procedure that reconfigures the FPGA. This is shown in the right branch of the tool flow of Fig. 2. The arguments of this procedure are the parameter values.

The fact that we model reconfiguration as a change of parameter inputs which can be expressed in the form of a high level HDL design and that we provide a method that automatically maps this design to a self-reconfiguring platform greatly relieves the designer of the burden to exploit the reconfiguration possibility at the LUT level.

We integrated the tool flow described in the previous paragraphs with the Xilinx XPS tool flow. The self-reconfiguring platform is implemented on a Xilinx Virtex-II Pro FPGA and the configuration manager is implemented on an embedded PowerPC (PPC) of the Xilinx device. The connection between the configuration manager and the configuration memory is realized through the Xilinx HWICAP module, which provides the interface between the OPB bus and the FPGA's ICAP (Internal Configuration Access Port). To configure parts of the FPGA fabric (LUTs) after a parameter value has changed, the PPC evaluates the tuning functions, generates the new configuration, and sends this new configuration to the FPGA configuration memory through the ICAP port of the FPGA via the HWICAP module. More details on our Xilinx tool flow instance can be found in [3].

4. Experiments

Experimental results on a 32-tap adaptive FIR filter, the filter coefficients are the parameters in this case, show that the use of self-reconfiguration with our tool flow improves the resource demands of the application by 40% [3]. Generating the new truth tables and reconfiguring the FPGA takes 151 ms in this implementation. In our current work we try to decrease the reconfiguration time even further.

5. Conclusion

Run-time hardware reconfiguration provides ample opportunities for optimizations of an implementation in time intervals in between two parameter changes. This paper provides a general tool flow that automatically maps any application that has a set of slowly varying inputs (called the parameters) to a self-reconfigurable platform. We have effectively integrated our tool flow in the Xilinx XPS tool flow that targets Virtex-II Pro FPGA devices. We used the embedded PowerPC of the Virtex-II Pro device as reconfiguration manager.

6. References

[1] K. Bruneel and D. Stroobandt, "Automatic generation of run-time parameterizable configurations", *in Proceedings of the International Conference on Field Programmable Logic and Applications*, 2008, pp. 361–366.

[2] K. Bruneel and D. Stroobandt, "Reconfigurability-Aware Structural Mapping for LUT-based FPGAs", *Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs*, 2008. pp. 223-228.

[3] K. Bruneel, F. Abouelella and D. Stroobandt, "Automatically Mapping Applications to a Self-reconfiguring Platform", *Proceedings of Design, Automation and Test in Europe*, 2009.

Acknowledgement

Karel Bruneel is supported by a B.O.F. grand of Ghent University. Ghent University is a member of the HiPEAC Network of Excellence.