Automating Reconfiguration Chain Generation for SRL-Based Run-Time Reconfiguration

Karel Heyse*, Brahim Al Farisi, Karel Bruneel, and Dirk Stroobandt

Ghent University, ELIS Department
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
{Karel.Heyse,Brahim.AlFarisi,Karel.Bruneel,Dirk.Stroobandt}@UGent.be

Abstract. Run-time reconfiguration (RTR) of FPGAs is mainly done using the configuration interface. However, for a certain group of designs, RTR using the shift register functionality of the LUTs is a much faster alternative than conventional RTR using the ICAP. This method requires the creation of reconfiguration chains connecting the run-time reconfigurable LUTs (SRL). In this paper, we develop and evaluate a method to generate these reconfiguration chains in an automated way so that their influence on the RTR design is minimised and the reconfiguration time is optimised. We do this by solving a constrained multiple travelling salesman problem (mTSP) based on the placement information of the run-time reconfigurable LUTs. An algorithm based on simulated annealing was developed to solve this new constrained mTSP. We show that using the proposed method, reconfiguration chains can be added with minimal influence on the clock frequency of the original design.

Keywords: FPGA, Run-Time Reconfiguration, Tuneable LUT Circuit, Shift-Register-LUT, SRL, Multiple Travelling Salesman Problem, Simulated Annealing

1 Introduction

Run-time reconfiguration (RTR) allows for more efficient utilisation of Field Programmable Gate Arrays (FPGA). Indeed, RTR enables us to specialise an FPGA's functionality for the current problem situation. This is called dynamic circuit specialisation (DCS) [1]. DCS can be done by simply writing a specialised configuration in the FPGA's configuration memory. A specialised configuration uses fewer resources and can attain higher clock speeds than a generic implementation and thus uses the FPGA resources more efficiently. The downside is that the gain in efficiency can be nullified by the reconfiguration overhead – the time needed to rewrite the configuration. Whether the dynamic circuit specialisation is useful depends on the problem itself, the rate at which the problem situation changes and the size of the reconfiguration overhead.

The reconfiguration overhead is greatly reduced when the TLUT method [1] is used to implement dynamic specialisation, because it only requires that some of

* Supported by a Ph.D. grant of the Flemish Fund for Scientific Research (FWO).

the FPGA's LUTs, called the TLUTs, are reconfigured in order to specialise the configuration of the FPGA and because the LUT truth tables constitute only ca. 4% (Virtex 2 Pro) of the complete configuration memory. Moreover, since only LUTs need to be reconfigured and not the routing, FPGAs that contain LUTs with shift register functionality (SRLs) offer the opportunity to perform even faster reconfiguration. Indeed, when a conventional reconfiguration port, like the ICAP, is used, many static configuration bits are rewritten because the ports require the reconfiguration of a full frame which contains many truth tables, some of which don't require reconfiguration. On the other hand, when using SRL reconfiguration [2], every LUT can be accessed individually which reduces the size of the configuration data that needs to be sent. Further, when using SRLs, all the LUTs can be reconfigured in parallel, while the reconfiguration ports of the FPGA have a restricted bandwidth. When this maximum reconfiguration bandwidth is not needed, the bandwidth can be customised to the design by arranging the SRLs as large shift registers, which we call reconfiguration chains.

The problem we address in this paper is how to introduce the reconfiguration chains so that their influence on the design is minimised. The only requirement is that each of the TLUTs is part of one of the reconfiguration chains in order to enable its reconfiguration. Which chain it is part of and where it is positioned in that chain are two degrees of freedom that can be used to minimise the impact the introduction of the chains has on the original design, and to optimise the reconfiguration time.

Our goal is to create an automatic and generic method to enable more widespread use of SRL reconfiguration, in particular in combination with the TLUT method but not restricted to it. This in contrast to previous work, which has mainly focussed on implementing individual problems using SRL reconfiguration.

We model the problem of generating these reconfiguration chains as a constrained multiple travelling salesman problem (mTSP) based on the placement information of the TLUTs after placement. A solution method based on simulated annealing is presented to solve this mTSP. We evaluated the proposed method by introducing reconfiguration chains into several dynamically specialisable FIR filter and TCAM designs, and found that for most problems the influence on the clock frequency of the design is limited to 5%, while allowing for extremely fast reconfiguration.

The paper is organised as follows. Section 2 gives a more profound introduction to the TLUT method and SRL reconfiguration. In Section 3 the problem is formulated as an mTSP and our proposed algorithm is described. Section 4 contains the experimental results. Finally, we draw conclusions in Section 5.

2 RTR Background

2.1 TLUT Method

A conventional FPGA tool flow transforms an HDL description of a design into an FPGA configuration. The TLUT method [1], on the other hand, is able to

construct a parameterised FPGA configuration, i.e. a dynamically specialisable configuration where some of the bits are expressed as Boolean functions of the parameters, from a parameterised HDL description. Particularly for the TLUT method, only the truth table bits of some of the LUTs are expressed in function of the parameters. All the routing, and thus the timing-information, of the design is fixed. The LUTs that have a parameterised truth table are called TLUTs.

Every time one of the parameters changes value, the new truth tables for the TLUTs are generated by evaluating the Boolean functions. The evaluation of these functions is performed by the configuration manager, which is generally implemented as software running on a (softcore) microprocessor [3]. Note that evaluating these Boolean functions is multiple orders of magnitude faster than generating the specialised configuration using a conventional FPGA tool flow.

2.2 SRL Reconfiguration

An efficient way to reconfigure the truth tables of the TLUTs is using SRLs (Shift Register LUTs) [2]. SRLs are LUTs whose truth table memory elements are also organised as a shift register (Fig. 1). Consequently, the truth table of an SRL can be changed by shifting data into the SRL. By repeatedly connecting the shift output of an SRL to the shift input of another SRL we can create longer shift registers (Fig. 2), which we call reconfiguration chains. When the reconfiguration chains are added to the design, the configuration manager can easily reconfigure all the TLUTs in the design by shifting the truth tables it has generated in the reconfiguration chains.[1]

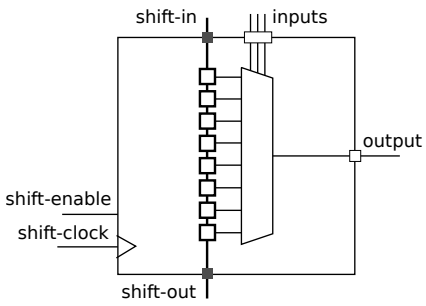


Fig. 1. A shift register LUT (SRL)

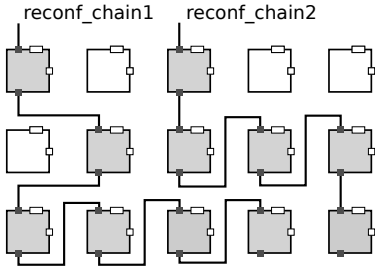


Fig. 2. Two reconfiguration chains connecting SRLs/TLUTs

Reconfiguration using SRLs is more efficient than using the ICAP for two reasons. First, using SRLs, only the truth tables of the TLUTs need to be written, while the ICAP always needs to rewrite a full frame.[2] A frame contains

[1] A reconfiguration clock tree and clock enable signal connecting all TLUTs are also added. They are controlled by the configuration manager.

[2] A frame is the atom of reconfiguration when the ICAP is used.

several truth tables, many of which are actually static. The bandwidth of the ICAP is thus used inefficiently. Second, when using SRL reconfiguration, the reconfiguration bandwidth can be adapted to the requirements of the system at hand, by choosing the number of reconfiguration chains. It has been shown in [4] that SRL reconfiguration can be up to two orders of magnitude faster than ICAP reconfiguration.

3 Automatic Linking of SRLs into Reconfiguration Chains

3.1 Problem Definition

In this paper, we automate the process of organising the TLUTs of an RTR design as a number of reconfiguration chains[3] so that their influence on the design is as low as possible and the reconfiguration speed is as high as possible.

Influence on the Design The order of the TLUTs in the reconfiguration chains and the distribution of all the TLUTs among the chains has an import influence on the total wire length of the extra connections required for the reconfiguration chains. Because they make use of the same programmable routing resources as regular connections, they influence the routability of the original design, and thus its clock frequency.

Reconfiguration Speed The reconfiguration time is defined as the product of the required number of clock cycles and the clock period of the reconfiguration clock. The number of reconfiguration clock cycles needed to reconfigure all the TLUTs is determined by the number of SRLs in the longest reconfiguration chain. This is because one clock cycle is needed for each configuration bit in a chain. Therefore, we need to minimise the number of SRLs in the longest chain. Since the total number of TLUTs is fixed and the number of chains is defined by the user, the lengths of the chains thus need to be balanced.

The clock period of the reconfiguration clock is approximately proportional to the length of the longest wire needed in the reconfiguration chains. We thus need to minimise the longest connection used to build the reconfiguration chains.

Optimising both the routability of the original design and the reconfiguration speed at the same time isn't always possible and sometimes requires a trade-off. Although in most situations minimising the total wire length will favour short connections, sometimes it is necessary to allow the total wire length to slightly increase to remove another long connection.

Modelling as a Multiple Travelling Salesman Problem The reconfiguration chains can be introduced at three points in the FPGA tool flow: before, during and after placement. We choose to add the reconfiguration chains after placement. At that point in the tool flow the position of every TLUT is known. This allows

[3] The number of chains is defined by the user.

us to model the introduction of reconfiguration chains as a constrained mTSP problem,[4] where not only the total traveling distance is minimised, but also the longest distance travelled between two cities and where the number of cities per salesman is balanced.

3.2 Previous Work on the Multiple Travelling Salesman Problem

The multiple travelling salesman problem has been extensively studied because of its many occurrences in real world problems. The main goal of all solution methods is to minimise the total path length. However, no previous research was found that also takes into account both the balancing requirement and minimisation of the longest connection.

k-opt k-opt is the name of a group of iterative solution methods for TSP, that was also adapted for mTSP [5]. The method does not take into account the balancing requirement or the length of the longest connection.

In each iteration of k-opt, k randomly chosen connections of a valid solution are broken and replaced with the k connections that minimise the total wire length. For small k, an exhaustive search for the new connections can be performed.

Solving a Balanced mTSP by Partitioning For the sake of testability, the flip-flops and registers in large ASIC designs are often organised in large shift registers, called scan chains. In [6] and [7] a method is proposed to introduce scan chains in two steps. In the first step the flip-flops are evenly partitioned based on their position relative to the position of the in and output pins of the scan chains. In the second step, each of the scan chains is interconnected using a regular travelling salesman solver.

We have considered this solution method for our work because of its straightforwardness and the similarity of the problem, but rejected it because of several reasons. First, the method of separating the partitioning and the solving of the individual TSP is not easily adaptable to take into account the longest connection. Depending on the design it might introduce unnecessary and very long connections. Secondly, the geometrical guidance used in these methods, the location of in and output pins, is not available to partition the TLUTs; only one common starting point, the configuration manager, and no endpoints are defined.

Solving an mTSP Using Simulated Annealing Simulated annealing is a heuristic that has been applied successfully to many problems in different domains, e.g. the placement problem in the FPGA tool flow. In [8] a theoretical

[4] The mTSP can be defined as follows: Given a collection of cities (the TLUTs) and a number of salesmen (the reconfiguration chains), find for each salesman a traveling route so that the combined traveling distance of all salesmen is minimised and each city is visited exactly once.

justification is given for applying the simulated annealing method (SA) to the mTSP problem. More useful information on the application of SA to the TSP problem can be found in [9].

The algorithm leaves a lot of freedom to the developer in defining the cost function to optimise. This is an advantage that distinguishes this method from those previously described, and it is the main reason we have chosen this technique for the proposed method.

3.3 Proposed Method for Solving the Constrained mTSP

The proposed solution method is based on simulated annealing. In our solution method, many SA implementation details are inspired by VPR's placement algorithm [10]. Some elements of the algorithm are based on 2-opt.

Simulated Annealing Simulated annealing is a heuristic to find the global minimum of a cost function inspired by the physical annealing of metals. It is an iterative algorithm in which a solution is repeatedly randomly altered and evaluated (Fig. 3). If a proposed alteration causes the cost of the solution to drop, the newly altered solution is accepted and used for further alterations in the next iterations. To avoid getting stuck in local minima, sometimes a solution with a higher cost is accepted as well. The probability of such a solution to get accepted depends on how much worse the solution is, and the value of the temperature (T) at the current iteration. The algorithm starts with a high temperature (high probability of acceptance) to allow exploration of the solution space. The temperature will then gradually decrease to make the solution converge to a minimum. The algorithm stops when the solution doesn't improve anymore. The temperature curve or annealing schedule is an important element of a simulated annealing based algorithm [10].

```
function simulatedAnnealing():
  s = s_0, T = T_0
  while not stopCondition
    repeat n times:
      c_previous = cost(s)
      s_proposed = randomAlteration(s)
      c_proposed = cost(s_proposed)
      Δc = c_proposed − c_previous
      if Δc < 0 or e^(−Δc/T) > random([0, 1])
        s = s_proposed
    T = nextTemperature(T)
  return s
```

Fig. 3. Pseudo code for a simulated annealing algorithm

Solution Space and Random Alterations Every TLUT was assigned a physical position on the FPGA during the placement step. We now define a second notion of position for the TLUTs, namely their position in the reconfiguration chains; each TLUT is assigned to one reconfiguration chain and has an index in this chain. When searching for efficient reconfiguration chains we will only change the positions of the TLUTs in the reconfiguration chains.

We start by randomly giving every TLUT a position in the reconfiguration chains. We only require that every reconfiguration chain has about the same number of TLUTs (at most 1 TLUT difference between the longest and shortest chain). In this way, the number of clock cycles to reconfigure all TLUTs is minimal. A single, common starting point, the configuration manager, is set for all the reconfiguration chains but the end points are not fixed.

Two alterations of a solution are possible. The first alteration (Fig. 4(a)) is based on 2-opt and was proposed in [9]. Two connections of a single reconfiguration chain are replaced in such a way that the new solution remains valid. The second alteration (Fig. 4(b)) is specific for the multiple travelling salesman problem and allows the interchange of TLUTs between the different reconfiguration chains. In this alteration two groups of TLUTs of equal size and from different reconfiguration chains are swapped.

Because both the alteration types will never change the number of TLUTs in a reconfiguration chain and the initial solution was balanced, the final solution will also be balanced.

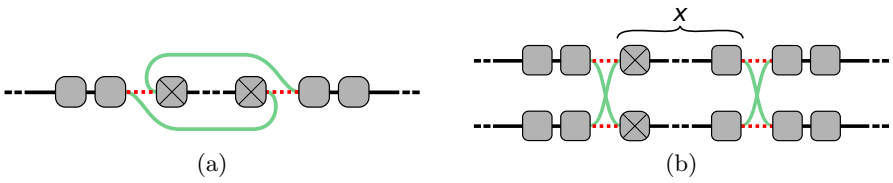


Fig. 4. Alterations of a solution. The dotted (red) lines are removed and replaced by the solid grey (green) lines. The rounded squares represent the TLUTs.

An alteration happens by randomly selecting a TLUT, and then selecting a second TLUT so that the Manhattan distance between the physical locations of the two TLUTs is not more than R. The variable R is used to keep the fraction of accepted solutions, β, close to $\beta_{target} = 35\%$ in order to speed up convergence [10]. R is adapted at each temperature iteration using the following equation:

$$R'_{new} = (1 - \beta_{target} + \beta) \cdot R_{old} \tag{1}$$

$$R_{new} = \max(R_{min}, \min(R'_{new}, R_{max})) \tag{2}$$

If the two randomly selected TLUTs (crossed squares in Fig. 4) already are on the same reconfiguration chain, then an alteration of type 1 is performed. If not, an alteration of the second type is used. The remaining degree of freedom,

the length of the group of TLUTs to exchange (represented by x in Fig. 4), is solved by exhaustively calculating the alteration with the lowest cost.

Cost Function The cost function used in the proposed algorithm is presented in Equation (3), where L represents a set of connections which form a valid solution for the reconfiguration chains, and l_i is the Manhattan distance between the physical positions on the FPGA of the TLUTs connected by i.

The first term of the cost function represents the total wire length of the reconfiguration chains. The second term is used to minimise the longest connection of the reconfiguration chains. The parameter α is introduced to make a trade-off between the two parts of the function possible.

Because of the restrictions on the initial solution and alterations, the solution will always be balanced and this requirement does not have to be included in the cost function; the number of clock cycles will always be minimal.

$$C_{SA} = \alpha \sum_{i \in L} l_i + (1 - \alpha) \cdot \sum_{i \in L} f(l_i) \tag{3}$$

$$f(l) = \begin{cases} l - 0.95 \cdot l_{max}, & \text{if } l > 0.95 \cdot l_{max} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$l_{max} = \max_{i \in L} l_i \tag{5}$$

The second term is an approximation of the real cost of the longest connection to speed up convergence to a solution with minimal longest connection length. We take a weighted sum of the lengths of all connections, where only the longest connections, $l > 0.95 \cdot l_{max}$, are given a penalty. This penalty is proportional to the length. An alteration that creates a connection that is longer than the currently longest wire will be penalised, while an incentive exists to replace existing long wires with shorter ones, even if they aren't the longest one.

At the end of the algorithm the weight function $f(l)$ is replaced by $f_{final}(l)$ (6). This removes the bias that is caused by penalising long connections which are not the longest connection ($0.95 \cdot l_{max} < l < l_{max}$). Those connections will have no influence on the longest connection so they can be used without a problem. The reason to apply this weight function only for finalisation is that it does not include an incentive to remove long connections, which is necessary to quickly reduce the length of the longest connection and make the solution converge.

$$f_{final}(l) = \begin{cases} \infty, & \text{if } l > l_{max} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Annealing Schedule An annealing schedule (Equation (7)), based on [10] with some minor experimentally determined differences, was chosen with exponential cooling using a variable parameter γ in function of the fraction of accepted alterations, β. The goal of the variable parameter $\gamma(\beta)$ is to make the algorithm

spend more time in the stage where the algorithm makes the most improvement, namely when β is between 5% and 96%.

$$T_{new} = \gamma(\beta) \cdot T_{old} \tag{7}$$

$$\gamma(\beta) = \begin{cases} 0.8, & \text{if } \beta \leq 5\% \\ 0.95, & \text{if } 5\% < \beta \leq 80\% \\ 0.9, & \text{if } 80\% < \beta \leq 96\% \\ 0.5, & \text{if } 96\% < \beta \end{cases} \tag{8}$$

4 Experimental Results

The proposed method was evaluated using two designs: a FIR filter (Finite Impulse Response filter) and a TCAM (Ternary Content Addressable Memory). The FIR filter has about 37% TLUTs while 60% of the TCAM's LUTs are TLUTs. For each design several instances of different sizes were used.

4.1 Results Using the VPR Tools

A first set of experiments evaluates the impact of the introduction of reconfiguration chains on the clock frequency of the design and was conducted with the commonly used academic VPR (Versatile Place and Route) tool version 5.02 [10]. The reconfiguration chains are generated based on the information from the VPR placer. Afterwards, when the reconfiguration chains have been added, routing is performed.

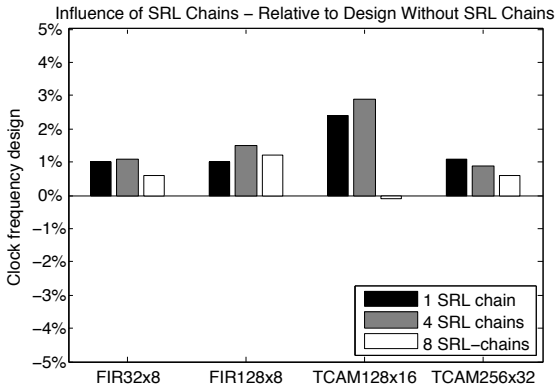


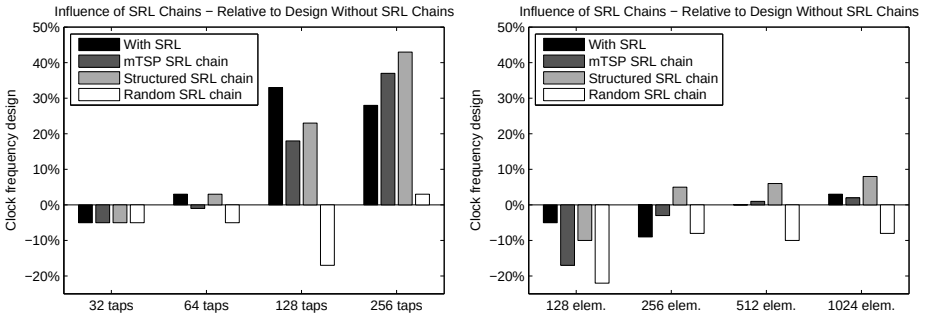
Fig. 5. Influence of the reconfiguration chains on the design's clock frequency using the VPR tool flow

From these experiments (Fig. 5) we conclude that, within the VPR tool flow and for the evaluated designs, it is possible to add up to (at least) 8 reconfigura-

tion chains with minimal negative influence on the design's clock frequency (less than 0.2%). In most cases we even notice a small improvement of up to 3%. This improvement is most likely caused by the addition of the reconfiguration clock tree, which influences placement. The frequency of the reconfiguration clock is between 4 and 24 times higher than the design's clock frequency.

4.2 Results Using the Xilinx Tools

A second set of experiments to evaluate the impact of the reconfiguration chains was performed with the commercial tools from Xilinx, primarily using the Virtex 2 Pro architecture. For these tests, the placement step was performed a second time after adding the reconfiguration chains. The reason for this is that the Xilinx tools do not easily allow editing a design after placement while retaining that placement. This might become possible in the future using recently developed tools such as Torc (`http://torc-isi.sourceforge.net`) or RapidSmith (`http://rapidsmith.sourceforge.net`).



(a) FIR filters of different sizes (b) TCAMs of different sizes

Fig. 6. Influence of the reconfiguration chains on the design's clock frequency using the Xilinx tool flow, averaged over experiments with 1, 4, 16 and 32 chains

Clock Frequency of the Design In Fig. 6 we see that for all but one problem instance (TCAM, 128 elem.), the clock frequency of the design, with the configuration chains added according to the proposed method, is at most 5% worse than the clock frequency of the design without reconfiguration chains, while in many cases we even notice an improvement. In comparison, when using randomly generated reconfiguration chains, the RTR design has a clock frequency that is up to 30% worse than the proposed method (FIR, 256 taps).

The improvement caused by adding the reconfiguration chains is mostly caused by the addition of the reconfiguration clock tree and reconfiguration enable signal and the conversion of regular LUTs into SRLs. This influences packing and placement of the TLUTs. The data labeled *"With SRL"* in Fig. 6 represents the clock speed of the design after adding just the reconfiguration

clock and reconfiguration enable signals, but no reconfiguration chains. We see that this influences the clock frequency significantly.

In some designs, a regular structure can be found, such as the memory elements of the TCAM or the taps of the FIR filter. In those designs, this structure can be used to (manually) determine an order of the TLUTs in the reconfiguration chains. It was found that this method (labeled *"Structured SRL chain"*) performed in some cases up to 10% better than the proposed method. This method however is not universally applicable.

Second Placement Step To determine the influence of the second placement step, which we are forced to do when using the Xilinx tools, we performed the experiment in VPR again using an additional placement step after adding the reconfiguration chains. The clock frequency of the design was 4% to 28% lower using this method. This is mainly because the replacement step shuffles the TLUTs which causes the mTSP problem to change and renders the previously calculated solution suboptimal. This may be counterintuitive because one expects the placer to place connected (T)LUTs closer together.

To illustrate this: we found that for the TCAM design, using the Xilinx tools, the total estimated wire length of the reconfiguration chains generated using the proposed method is on average 130% longer after the second placement step than before.

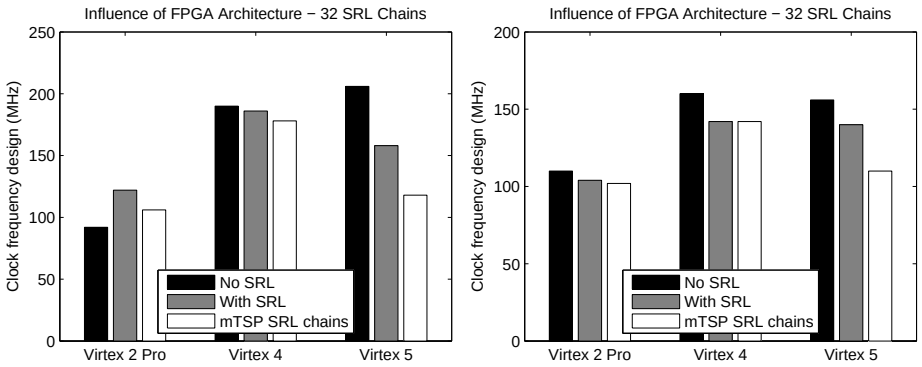
Number of Reconfiguration Chains The influence of the number of reconfiguration chains (not in the figures) on the design's clock is negligible for up to 32 chains, while the influence on the reconfiguration clock is small enough to allow faster RTR by using parallel reconfiguration. The frequency of the reconfiguration clock ranges from 1x to 2x the frequency of the design's clock.

FPGA Family Additional experiments were performed on the Virtex 4 and Virtex 5 families (Fig. 7). For large designs with large numbers of TLUTs a significant degradation of the design's clock frequency was noticed using the Virtex 5. The reason for this is that in the Virtex 5 only 25% of all LUTs can be configured as SRLs, and therefore the placement of the design can be worse. A similar deterioration can be found for the Virtex 4, which has 50% SRLs, but only for the TCAM design which has 60% TLUTs and not for the FIR filter with only 37% TLUTs. The limited number of SRLs can make it impossible to implement large designs with a high fraction of TLUTs on these families of FPGAs.

5 Conclusion

An automated solution method based on simulated annealing was proposed to generate reconfiguration chains, which are needed to perform SRL reconfiguration. When generating the reconfiguration chains, our method takes into account the routability of the original design and the reconfiguration time.

Experiments using the VPR and Xilinx tool flows show that, when adding up to 32 reconfiguration chains using the proposed method, the influence on



(a) FIR filters, 128 taps (b) TCAM, 128 elements

Fig. 7. Influence of the FPGA family on the design's clock frequency when using 32 reconfiguration chains

the design's clock speed is in most cases limited to 5%. The quality of the reconfiguration chains generated by the proposed method approaches that of a hand made solution and is significantly better than that of a random solution.

Currently the Xilinx tools force us to perform a second placement step after adding the reconfiguration chains. Experiments show that our results could still be improved if a technique was implemented to avoid this.

References

1. Bruneel, K., Heirman, W., Stroobandt, D.: Dynamic Data Folding with Parametrizable FPGA Configurations. In: ACM TODAES, vol. 16, pp. 43:1–43:29. (2011)
2. Dynamic Constant Coefficient Multiplier v2.0. Xilinx. (2000)
3. Abouelella, F., Bruneel, K., Stroobandt, D.: Towards a More Efficient Run-Time FPGA Configuration Generation. In: Proceedings of ParCo '09, pp. 113–116.
4. Al Farisi, B., Bruneel, K., Devos, H., Stroobandt, D.: Automatic Tool Flow for Shift-Register-LUT Reconfiguration. In: Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 287. (2010)
5. Potvin, J., Lapalme, G., Rousseau, J.: A Generalized K-Opt Exchange Procedure for the mTSP. In: INFOR, 27, pp. 474-481. (1989)
6. Kobayashi, S., Edahiro, M., Kubo, M.: A VLSI Scan-Chain Optimization Algorithm for Multiple Scan-Paths. In: IEICE Trans. Fund., vol. E82-A, pp. 2499–2504. (1999)
7. Rahimi, K., Soma, M.: Layout Driven Synthesis of Multiple Scan Chains. In: IEEE TCAD, vol. 22, pp. 317–326. (2003)
8. Song, C., Lee, K., Lee, W.D.: Extended Simulated Annealing for Augmented TSP and Multi-salesmen TSP. In: Proceedings of IJCNN '03, vol. 3, pp. 2340–2343.
9. Kirkpatrick, S.: Optimization by Simulated Annealing: Quantitative Studies. In: Journal of Statistical Physics, vol. 34, pp. 975–986. (1984)
10. Betz, V., Rose, J., Marquardt, A.: Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, Norwell (1999)