A Reconfiguration Aware Circuit Mapper for FPGAs

Markus Rullmann and Renate Merker
Dresden University of Technology, Germany
Circuits and Systems Laboratory

E-mail: `markus.rullmann@tu-dresden.de`

Abstract

Dynamic reconfiguration for fine grained architectures is still associated with significant reconfiguration costs. In this paper we propose a new reconfiguration aware design flow. The tools in this flow implement a set of tasks concurrently. The flow leads to task implementations with minimal costs for routing reconfiguration. This is mainly achieved by our mapping tool which solves two fundamental problems: Our mapping algorithm generates variants for the mapping of netlist cells to logic blocks. From those logic blocks a subset for each task is selected that minimizes the cost for routing reconfiguration. We derive a cost function and formulate an integer linear program to solve this problem. We implemented several task sets with our method and compare the results to previous solutions. We show that the reconfiguration aware mapping leads to better results than early approaches with vendor provided tools.

1 Introduction

Dynamic reconfiguration for FPGA architectures is an established design method to increase the efficiency of system designs. Sequential tasks are reconfigured at runtime to reduce the resources usage. However, current vendor design tools – device mapper and place and route tools – are not aware of the relation between tasks and their implementation. Hence, they can not optimize the task configurations to reduce reconfiguration costs.

In [7] we established a method to extract structural similarities in tasks. We have further shown how this information can be used to implement tasks with reduced reconfiguration costs [6]. Unfortunately the standard implementations tools have some shortcomings: only the relation between two tasks benefits directly from the flow and tasks are not treated concurrently, information can only be transfered from one completely implemented design to the design currently in implementation.

In this paper we present a new reconfiguration aware mapping tool: The tool performs the mapping of all tasks in a set simultaneously. Therefore, the tool investigates a variety of mapping alternatives for each task and selects one implementation for each task that minimizes the reconfiguration costs between all tasks in a set.

1.1 Reconfiguration Aware Design Flow for FPGAs

In an earlier work [6], a *Matched Implementation Flow* was described and deficiencies associated to the use of the vendor implementation tools were highlighted. The work inspired us to create a new *Reconfiguration Aware Design Flow* for FPGAs. The starting point is a given task set. Each task is given by a netlist generated with a logic synthesis tool (e.g. in edif format), which consists of cells and nets. At first, the similarities between those tasks are extracted using the methods in [7]. The similarities are described by *matching cells* and matching connections between those matching cells. This information is used in the subsequent implementation steps. The device mapper translates the synthesis netlist into a device specific *map netlist.* In the map netlist, one or more cells are merged into an appropriate logic block (LB). This process is called packing. Our mapper will create a netlist that allows the place and route tool to have minimal routing reconfiguration for the given tasks. For this purpose it also needs to translate the information on matching cells to *matching LBs.* We will show that this is a computationally non-trivial problem. The algorithms and objectives of the mapper are

1-4244-0910-1/07/$20.00 ©2007 IEEE

described in detail in this paper, see Sections 2 and 3. The placement tool places LBs at physical resources in the device. In order to minimize reconfiguration costs, matching LBs are placed at the same resources for relevant tasks. This ensures that the routing tool can take advantage of the matching connections and route them using the same switch block configuration in two or more tasks.

The tool flow can significantly reduce the amount on reconfiguration costs for a given set of tasks. We developed tools for task similarity analysis, mapping, and a preliminary version for the placement tool. The analysis of the remaining reconfiguration costs gives promising results, details are given in Section 5.

1.2 Architecture Model

The most common architectures today are island style FPGAs. These FPGAs have LBs of different functionality, e.g. general logic, I/O blocks, dedicated multipliers, RAM blocks etc. The LBs are connected to a generic routing architecture, consisting of wires and configurable switch blocks. We assume that the switch blocks require much more data to configure routing structures than the LBs require data to configure their functionality. This is true e.g. in Xilinx VirtexII architectures. We found that for each CLB column 22 frames are required for configuration – with 18 out of 22 frames used for configuring the switch blocks. In the Xilinx Virtex and Spartan architectures frames can be written selectively, which allows to reconfigure only parts of the device that require different configurations in the tasks.

The organization of configuration data in frames imposes a restriction on placement and routing of tasks: if one or more bits in a frame differ between the tasks' configurations, the whole frame must be reconfigured, regardless of the number of different configuration bits. The presented circuit mapper aims at generating a map netlist that is an optimal input to a reconfiguration aware place and route tool. Hence, the resulting netlist has the least possible difference in LB connections. In the next step, the place and route tool must arrange the placement such that the router can generate switch block configurations with the least possible difference in configuration frames.

Our approach dissolves the difference between static and dynamic functionality in a task set. It is replaced by the notion similarity between the netlists. In case of similar LBs and connections the place and route tools can treat these structures as static portions, while the differing parts are object to place and route with optimal frame difference.

2 General Mapping Approach

Generally, a mapping tool binds the cells given in a synthesis netlist to resources of LBs in the FPGA target architecture. In Figure 1 a simple circuit and some *mapping variants* are shown. The circuit — given as a synthesis netlist (Figure 1(a)) — consists of cells and nets. The cells represent the circuit function and the nets the connections between those functions. The mapping tool usually has many possibilities to assign the cells from a netlist to resources inside LBs (Figure 1(b,c)). Mapping also includes logic packing to place multiple cells inside one LB, see Figure 1(b). Usually, packing is driven by three main objectives:

- propagation delay between gates
- task routability
- minimal resource usage.

Algorithms targeting the above objectives are described e.g. in [2].

The nets given in the synthesis netlist and the mapped cells in the LBs determine which output pins of a LB must be connected with the input pins of other LBs. The nets are realized as signal routes in the FPGA fabric. Using logic packing, a net can become local to the LB (Figure 1(b), N1) and thus does not utilize the routing architecture. Local nets are often less costly to reconfigure. Packing can also cause merged net pins (Figure 1(b), N2). Instead of routing all connections from the net driver to each load in the netlist individually, merged net pins require only one connection to the LB pin (Figure 1(b), (3)) – the connections to each cell pin are local to the LB.

Todays FPGAs have increasingly complex LB architectures with support for shift registers, carry chain logic, large multiplexers etc. This often requires packing of specific cells to ensure routability, since not all resource pins in a LB are connected to the switch blocks. In this paper we present a mapping algorithm which automatically packs cells that must be connected with local routes in one LB. Thus we avoid special design rules for describing packing.

2.1 Mapping Variants and Reconfiguration Costs

The mapping variant chosen by the device mapper has a direct impact on the routing configuration. As discussed before, a cell can be mapped to different resources in a LB. Consequently, the nets will be routed using different LB pins. However in our tool flow we assume that matching edges in the tasks can be routed using the same switch block configuration, hence the

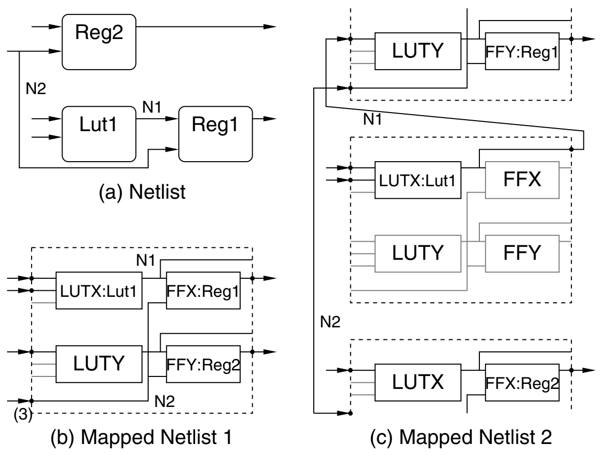

Figure 1: A simple circuit and some mapping variants

same routes, in the FPGA. One objective of the mapper is to select the mapping variants that map matching connections to the same LB pins for matching cells in the given tasks.

Consider the example given in Figure 2: We assume a given mapping for the cells A–C in task 1. The cells A–C in task 2 shall match the corresponding cells in task 1. The architecture allows two mapping variants for cell C in task 2. Obviously, the routing of matching routes (bold lines) is static if variant 2(b) is chosen meanwhile variant 2(a) causes routing reconfiguration for similar connections.

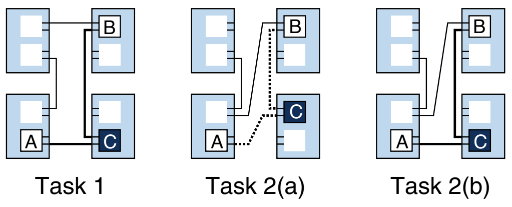

Figure 2: A given mapping for task 1 and possible mapping variants for task 2. Bold lines are matching edges in the cell netlist. Depending on the chosen mapping of cell C (task 2(a) or 2(b)), some routes require reconfiguration (task 2(a)) or can remain static (task 2(b)).

2.2 Logic Block Generation

In this section we describe our approach for a mapping algorithm that automatically maps cells to LB resources and packs the cells so that first routability inside the LB and second the mapping of locally connected cells to a single LB is enforced. The major advantage of the algorithm is the independence from any specific target architecture. The mapping algorithm is described in Figure 3 and works as follows:

```
1   for all $c \in \mathcal{C}_i$
2    for all $m_c \in \mathcal{M}_c$
3     initialize empty black_list, routable_list
4     initialize edif_queue with $c$ as initial element
5     create empty LB $l$
6     while edif_queue not empty
7      $c_k$ := front( edif_queue)
8      for all $m_{c_k} \in \mathcal{M}_{c_k}$
9       if ((($c = c_k$ and $m_c = m_{c_k}$) or $c! = c_k$)...
          and lb_type( $m_c$) = lb_type( $m_{c_k}$))
10        $l' := l$
11        ok = map $c_k$ to $l'$ according to the rules $m_{c_k}$
12        if ( ok and $c_k$ adds new local routes) or $c = c_k$
13         $l := l'$
14         if $l$ routable
15          add $l$ to routable_list
16         exit loop (8)
17     if $c_k$ was mapped to $l$
18      add edif instances connected with $c_k$ ...
        to back of edif_queue
19     else
20      add $c_k$ to black_list
21     remove $c_k$ from edif_queue
22    add front( routable_list) and back( routable_list)
  to $L_i$
```

Figure 3: Pseudo code for the generic LB generation algorithm

Let us assume that the netlist of task i comprises a set $\mathcal{C}_i$ of cells c. There is a set $\mathcal{M}_c$ of variants m_c to map cell c to LB resources. We will generate a set $\mathcal{L}_i$ of LBs l such that connected cells in the netlist are implemented on a single LB and routability inside the LBs is ensured.

First, for each $c \in \mathcal{C}_i$ (1) the set $\mathcal{M}_c$ of possible mappings to resources of a LB (2) are determined by using a greedy algorithm. Two lists (3) and a queue (4) are introduced for controlling the following operations. The mapping procedure starts from one mapping version $m_c \in \mathcal{M}_c$ of cell $c \in \mathcal{C}_i$ to LB resources, where c is called initial cell. For this case, the edif_queue contains all cells connected to the initial cell c. For each cell c_k in the edif_queue, all possible mapping variants $m_{c_k} \in \mathcal{M}_{c_k}$ to resources of a current LB l are investigated in sequential (8), until a valid mapping for each cell is found. Condition (9) determines which mappings for c_k are investigated: For the initial cell, the only mapping m_c is allowed; For all other cells, any mapping to the same type of LB as for m_c can be chosen. A mapping is valid, if the LB has free resources to map the cell according to m_{c_k} – and – if the mapping leads to any new route *inside* the LB (12). Condition (12) also skips the local route condition for the initial cell c mapped to the LB resources as well. After this mapping procedure the LB l is generated. If the LB l is routable, it is stored temporarily in routable_list. In line (18), we add all cells that are loads or drivers of c_k to the back of edif_queue. If one cell of the edif_queue could not be mapped to resources of the current LB, it is inserted into black_list (20), a list of cells that will not be inserted into edif_queue again.

If loop (6) is finished, the first and last element of routable_list are added to the set $\mathcal{L}_i$ of generated LBs.

This procedure is repeated for all cells $c \in \mathcal{C}_i$ and their corresponding mapping versions $m_c \in \mathcal{M}_c$.

The result of the mapping algorithm can be summarized in a relation $\mathcal{R}$ for task i, where the elements (c_j, l_k) describe the mapping of cell $c_j \in \mathcal{C}_i$ to the generated LB $l_k \in \mathcal{L}_i$.

The presented algorithm generates all mapping variants for each initial cell and ensures routability of the generated LBs by packing all cells that lead to local routes. However, the logic packing is only performed in a greedy manner, hence there is only one feasable mapping variant investigated for each cell connected with the initial cell. This reduces the amount of LBs generated but results still in a variety of different LBs for each cell. Note that this algorithm does not perform packing of unconnected cells, this separate problem is discussed e.g. in [5].

2.3 Logic Block Selection

In this section, all functions $\mathcal{A} \subset \mathcal{R}$ will be selected for task i, where $\mathcal{L}_{mi}$ denotes the range of $\mathcal{A}$. The functions $\mathcal{A}$ assign each cell $c_i \in \mathcal{C}_i$ exactly one LB. We give an integer linear program (ILP) formulation of the problem that will be extended in Section 3 to select optimal LBs for reconfiguration. The ILP formulation of this problem is straightforward. For each LB l_i, we define a binary variable S_{li} that is 1 if the LB l_i is element of the range of a function $\mathcal{A}$ or 0 if not. To select a function the following constraint for each cell c_i is used:

$$1 = \sum_{l_i \in \mathcal{L}_{ic}} S_{li},$$

where the subset $\mathcal{L}_{ic} \subset \mathcal{L}_i$ of generated LBs denotes the set of LBs implementing cell c_i.

3 Reconfiguration Aware Logic Block Selection

In this section we extend our LB selection problem to optimize reconfiguration costs. The objective is to reduce the number of routed nets that need to be reconfigured between tasks, since routing configuration is the major cost factor in fine grained architectures.

3.1 Cost Function: Minimal Route Reconfiguration

The synthesis netlists and the map netlists are modelled as a digraph here. In the following, we distinguish edges from the synthesis and map netlists by *net edges* and *route edges*, respectively. A net edge can occur as route edge in the map netlist as follows:

- a LB local route edge, not being routed in the fabric but inside the LB only
- a LB external route edge, being routed from/to LB pins through the FPGA fabric

Due to logic packing it is also possible that more than one net edges are merged into a single route edge. The cells are then connected internally to the same LB pin. For task i the number E_i of net edges in the synthesis netlist translates to:

$$E_i = R_i^{\text{local}} + R_i^{\text{external}} + R_i^{\text{merged}}.$$

The number E_i of net edges is constant, while the number of local route edges R_i^{local}, external route edges R_i^{external} and merged route edges R_i^{merged} depends on the selected LBs for task i. The external route edges can be divided into matched and unmatched route edges, both depending on the LB selection in task i and task j:

$$R_i^{\text{external}} = R_{ij}^{\text{matched}} + R_{ij}^{\text{unmatched}}$$

The LBs must be selected such that the number of unmatched route edges is minimized, which reduces costs for routing reconfiguration. Since E_i is constant, the problem is equivalent to maximizing the other terms:

$$\min R_{ij}^{\text{unmatched}} \mathrel{\widehat{=}} \max\,(R_{ij}^{\text{matched}} + R_i^{\text{local}} + R_i^{\text{merged}}). \tag{1}$$

The terms R_i^{local} and R_i^{merged} can be directly calculated from the LB selection for each task i:

$$R_i^{\text{local}} = \sum_{l_i \in \mathcal{L}_i} o_{li}^{\text{local}} S_{li}$$

and

$$R_i^{\text{merged}} = \sum_{l_i \in \mathcal{L}_i} o_{li}^{\text{merged}} S_{li}$$

with o_{li}^{local} is equal to the number of local connections inside a LB and o_{li}^{merged} the number of LB internal pins using the same LB external pin.

However the term $R_{ij}^{\text{unmatched}}$ describes only the reconfiguration costs when replacing task j by task i. In the general case, reconfiguration can occur between a set of I different tasks. In that case the objective function is given by:

$$\min \sum_{i \in \mathcal{I}} \sum_{j \neq i\, j \in \mathcal{I}} R_{ij}^{\text{unmatched}}$$

The number of matched route edges is symmetric, hence $R_{ij}^{\text{matched}} = R_{ji}^{\text{matched}}$ so the objective function can be written as:

$$\max\,[2 \sum_{i \in \mathcal{I}} \sum_{j < i\, j \in \mathcal{I}} R_{ij}^{\text{matched}} + (I-1) \sum_{i \in \mathcal{I}} (R_i^{\text{local}} + R_i^{\text{merged}})].$$

Observe that, in order to minimize routing reconfiguration we can maximize the use of LB internal routing and/or increase matching route edges between the tasks.

3.2 Logic Block Matching

The similarity of tasks can be given in terms of matching cells [7]. Each cell in a task may have one matching cell in one or more other tasks. For task i and task j matching cells are selected such that as many connecting net edges as possible between the cells in task i can also be found between the matching cells in task j. Consider the example in Figure 4: (c) shows matching cells for the synthesis netlists in (a,b).

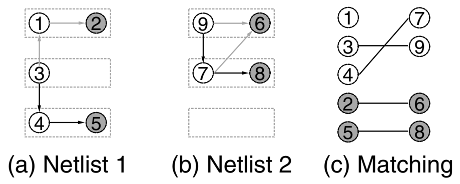

Figure 4: Example for matching cells: (a), (b) cell netlists with cells (circles) mapped to LBs (boxes). Matching net edges are drawn black. (c) cells connected are matching cells.

The mapping process must translate the information on matching cells to matching LBs. Matching LBs are placed at the same physical site by our placement tool. The route edges that describe the same connections between LBs in tasks i, j can now be routed using the same switch box configuration in those tasks. Thus reconfiguration costs can be reduced.

The LB generation algorithm treats all tasks independently, without matching information. Each LB of task i contains one or more cells. The cells can have matching cells in another task j. Since the LBs in task j are generated independently from task i, the matching cells are not necessarily packed into a single LB as in task i. I.e. in Figure 4 the cells (9,6) are packed to one LB in netlist 2 but the matching counterparts (3,2) in netlist 2 are mapped to separate LBs. The purpose of LB matching is to find those LBs that retain most of the structural similarity defined by the matching cells. The conditions for LB matching are described more formally in the following.

A set $\mathcal{M}_{i,j}^{C} = \{\ldots, (c_{ni}, c_{nj}), \ldots \mid c_{ni} \in \mathcal{C}_i\,, c_{nj} \in \mathcal{C}_j\}$ of matching cells is given for the tasks i, j, see [7]. Since every cell c_{ni} can initially mapped to any LB in $\mathcal{L}_{ic}$ and c_{nj} to any LB in $\mathcal{L}_{jc}$ respectively, the set $\mathcal{M}^{\text{LB}}$ of possible matching LBs is given by:

$$\begin{aligned} \mathcal{M}^{\text{LB}} &= \bigcup_{(c_i, c_j) \in \mathcal{M}_{i,j}^{C}} \mathcal{M}_{c_i c_j}. \\ \text{with } \mathcal{M}_{c_i c_j} &= \mathcal{L}_{ic} \times \mathcal{L}_{jc} \end{aligned}$$

The solution to the LB matching problem is a subset $\mathcal{M}^{\text{LB},m} \subset \mathcal{M}^{\text{LB}}$ that fulfills the following conditions:

- $l_i \in \mathcal{L}_{mi}$ and $l_j \in \mathcal{L}_{mj}$
- for each l_i there exists not more than one $(l_i, l_j) \in \mathcal{M}^{\text{LB},m}$ and vice versa

In the ILP we introduce a binary variable $S_{li,lj}$ that is set to one if $(l_i, l_j) \in \mathcal{M}^{\text{LB},m}$. The following constraints ensure the conditions stated above:

$$S_{li} \geq \sum_{lj \in L_{mj}} S_{li,lj} \qquad (2)$$

$$S_{lj} \geq \sum_{li \in L_{mi}} S_{li,lj} \qquad (3)$$

3.3 Route Edge Matching

The matching route edges are given by the LB matching. However, to formulate the cost function completely, it is required to evaluate the number of matching route edges for any set of matching LBs. In [7] we extracted the edge matching from completely matched graphs. Here we introduce the notion of matching pairs in order to define constrains for the ILP program.

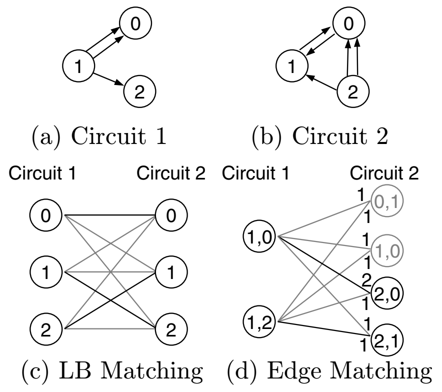

Figure 5: LB matching example: (a,b) example map netlists, (c) matching for LBs, (d) graph showing possible matching pairs and the number of possible route edge matches. Black edges in (c,d) highlight a feasible solution.

We investigate the route edges between each pair of LB (l_{1i}, l_{2i}) in task i and compare those route edges to any pair of matching LBs (l_{1j}, l_{2j}) in task j. The number of matching route edges is given by $o^{\text{matched}}_{l_1 i,l_2 i,l_1 j,l_2 j}$.

The LB matching determines the LBs that must be placed at the same physical resource during placement. Consequently, route edges that connect equal pins of matching LBs in the given tasks are matching route edges.

Consider the example in Figure 5: given are the circuits 1 and 2. From all possible matching LBs, we select $\mathcal{M}^{\text{LB},m} = \{(0,0),(1,2),(2,1)\}$ denoted by black edges in Figure 5(c). In Figure 5(d) all pairs of LBs with at least one route edge are given as nodes for each circuit separately. The number of matching route edges for all possible matching pairs is annotated to the edges of the graph in Figure 5(d). In correlation with the given matching in Figure 5(c), the resulting matching pairs are: $((1,0),(2,0))$ and $((1,2),(2,1))$, thus leading to a total of 3 matching route edges.

In the ILP we introduce another binary variable that holds the information about matching pairs. $S_{l_1 i,l_2 i,l_1 j,l_2 j}$ shall be one if $(l_{1i}, l_{1j}),(l_{2i}, l_{2j}) \in \mathcal{M}^{\text{LB},m}$. Therefore the following constraints must be true:

$$S_{l_1 i,l_1 j} \geq S_{l_1 i,l_2 i,l_1 j,l_2 j} \tag{4}$$

$$S_{l_2 i,l_2 j} \geq S_{l_1 i,l_2 i,l_1 j,l_2 j} \tag{5}$$

The ILP formulation is concluded with the cost term for matched routes:

$$R^{\text{matched}}_{ij} = \sum_{(l_{1i},l_{1j}),(l_{2i},l_{2j}) \in \mathcal{M}^{\text{LB},m}} o^{\text{matched}}_{l_1 i,l_2 i,l_1 j,l_2 j} S_{l_1 i,l_2 i,l_1 j,l_2 j}.$$

4 Reconfiguration Aware Mapping Tool

We integrated the methods described in Sections 2 and 3 into a unique reconfiguration aware mapping tool. The tool maps simultaneously a complete task set and takes advantage of their structural similarity. This approach can be seen complementary to the commercial tools like Xilinx PlanAhead [8] and Atmel Figaro (for FPSLIC Devices) [1]. PlanAhead supports only physical design partitioning and module management. Figaro does only support bitstream compression by treating different tasks as contexts. Design partitioning into static and reconfigurable areas must be solved by the designer. Our tool focuses on the reconfiguration aware design flow that reduces reconfiguration costs by mapping, placement and routing strategies.

Our tool acts in four main stages: at first, the formal architecture description, the mapping rules and the synthesis netlists are read in; secondly, it generates the set of LBs for the designs independently according to the algorithm in Section 2; from those, the tool selects the LBs for each task, as well as the LB matching that guarantees minimum routing reconfiguration between the given tasks. This is achieved by automatically generating the ILP as described in Section 3 and using the commercial ILP solver Ilog CPLEX. At last, the mapped tasks are written to separate map netlists files. The tool also produces a file that describes the matching of LBs and route edges, which is used as an input to the place and route tool.

4.1 Interface to FPGA Vendor Tools

Currently, our tool supports data files that are compatible to Xilinx ISE design flow. Hence, the steps in the ISE flow can be replaced selectively by our tools. The synthesis netlist is read in edif format, which is also supported by other commercial tools. The map netlist is written in the proprietary Xilinx description language (.xdl), which can be converted to the binary .ncd format. However, the internal representation in our tools is not dependent on Xilinx devices and other file formats could be easily adapted.

4.2 Mapping Rules

In our approach, the architecture description is separated from the mapping rules, which enables the tool to read the vendor provided architecture description[1]

[1]e.g. generated with Xilinx xdl

and the mapping rules independently. The architecture description consists of definitions for all types of LBs (e.g. I/O blocks, LUT logic and function macros) supported by the device. Each LB type is composed of discrete functional elements, configurable multiplexers and LB internal wires. The mapping rules describe all binding variants for the cells to resources in a LB. A routing subroutine in the mapping tool can automatically route nets to LB pins using wire and configurable multiplexer definitions taken from the architecture description.

5 Experimental Results

We run our reconfiguration aware mapping tool on a number of example tasks. We used a set of tasks (b*xx*) from the itc'99 benchmarks [4]. Tasks add8, sub8 contain an 8bit add and subtract circuit with registered outputs, respectively. Tasks int, motion are introduced in [6]. Tasks opb_add and opb_sub are also reconfigurable IP cores, see [3]. The task sets have been analyzed with our matching tool.

In Table 1, we run the mapping tool for each task separately, without LB matching. The results show the number of cells in the synthesis netlist and the number of generated LBs. On average, there are two valid mapping variants for each cell. The results show that about one third of the generated LBs form the set of selected LBs. The solution to the LB selection problem requires only small execution time when compared to the reconfiguration aware LB selection.

Task	Cells	Generated LB	Selected LB
b01	30	70	21
b02	14	30	10
b06	29	62	21
b03	174	424	141
b08	95	218	74
b10	119	265	95
add8	58	71	32
sub8	60	74	33
int	932	1629	555
motion	985	1801	578
opb_add	300	614	262
opb_sub	300	614	262

Table 1: Mapping results from LB generation and LB selection

The results for the task sets are given in Table 2. Task set 1 and set 2 contain similar sized netlists from the itc'99 benchmark suite. The tasks in these task sets have very little similarity. The purpose is to demonstrate our method on task sets with more than two tasks. The task sets 3–5 have a very similar structure and are ideal test cases for our method. In Table 2 the results for net edges, route edges LBs and pins are shown for the task sets. The numbers given are the sum of edges, pins and LBs for all tasks in each set. For each parameter, the total and the amount of equal entities is given. The difference between total and equal edges or pins is a quantitative measure for the amount of routing that requires reconfiguration. Task set 6 contains data derived from the Xilinx implementation flow as it has been used in [6]. The differences between the results for set 5 and set 6 highlight the advantages of our reconfiguration aware design flow: from the same netlists, our mapping tool creates a map netlist with fewer LB pins, which requires less routing for the tasks. Furthermore, the difference between total pins and equal pins is much less for set 5, which means there are less routes to be reconfigured between those tasks. An interesting fact is that the Xilinx tool performs a more aggressive packing and maps the cells to considerable less LBs. Nevertheless, the matched implementation flow leads to map netlists with many more total and different pins which causes more reconfiguration overhead when compared to our approach.

6 Conclusion

In this paper we presented a new reconfiguration aware design flow. The major objective of the design flow is an implementation of tasks with minimal reconfiguration costs. The tools achieve minimum costs by treating the given tasks concurrently in contrast to existing approaches. We described the mapping tool in detail. The tool constructs a set of mapping variants for each task. From those variants, the tool selects one for each task than minimizes reconfiguration costs between all tasks in the set.

In our experiments we have shown the application to relevant tasks, such as reconfigurable hardware accelerators. We have shown that our tool implements tasks with less overhead than previous solutions.

Currently, the mapper does not perform logic packing of unconnected cells. Hence, only logic that leads to a local route inside an LB is packed. The results often in low utilization of the logic inside an LB. However, we believe the results can be improved by adding suitable packing heuristics that result in more local or merged route edges. The mapper can be extended by established algorithms, e.g. [5], to perform more aggressive packing. However the proposed reconfiguration aware logic block selection can still be applied.

Our mapper generates a number of mapping variants for each initial cell and adds connected cells to the LB

Task Set	Tasks	Net Edges			Route Edges			LBs	Pins		
		total	equal	diff.	total	equal	diff.		total	equal	diff.
set1	b01, b02, b06	184	72	112	170	47	123	52	234	58	176
set2	b03, b08, b10	1211	378	833	1173	213	960	310	1547	224	1323
set3	add8, sub8	220	216	4	92	88	4	65	171	168	3
set4[2]	opb_add, opb_sub	1938	1402	536	1374	1268	106	524	2342	1716	626
set5[2]	int, motion	7279	5598	1681	4586	3708	878	1133	6511	4120	2391
set6[3]	int, motion							776	7293	4594	2699

[2]ILP optimization aborted after a valid solution was found.

[3]Results from Matched Implementation Flow using Xilinx guide mode, see [6].

Table 2: Reconfiguration aware map – results

chosing one variant only. We believe that it is not reasonable to provide all mapping variants of connected cells, even if the algorithm would allow it with minor modifications. The huge increase in mapping variants for each cell could render to the logic block selection problem hardly solvable. One approach might be to implement a problem specific optimization algorithm in this case.

In the future we plan to complete the placement tool and to develop a reconfiguration aware router. These tools will enable us to compare reconfiguration costs at bitstream level with implementations from vendor tools.

References

[1] Atmel Corp. Figaro: www.atmel.com/products/fpslic.

[2] V. Betz and J. Rose. VPR:a new packing, placement and routing tool for fpga research. In *Int. Workshop on Field-Programmable Logic and Applications*, pages 213–222, 1997.

[3] C. Claus, F. Müller, and W. Stechele. Combitgen: A new approach for creating partial bitstreams in virtex-II pro devices. In *International Conference on Architecture of Computing Systems – Workshop on Dynamically Reconfigurable Systems, ARCS 2006*, GI Lecture Notes in Informatics, pages 122–131, March 2006.

[4] F. Corno, M. S. Reorda, and G. Squillero. RT-Level ITC99 benchmarks and first ATPG results. *IEEE Design and Test of Computers*, pages 44–53, July–August 2000.

[5] A. Marquardt, V. Betz, and J. Rose. Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 37–46, Monterey, CA, February 1999.

[6] M. Rullmann and R. Merker. Design and implementation of reconfigurable tasks with minimum reconfiguration overhead. In *Dynamically Reconfigurable Architectures Workshop at 19th International Conference Architecture of Computing Systems (ARCS 2006)*, pages 132–141, Frankfurt/Main, Germany, March 2006.

[7] M. Rullmann and R. Merker. Maximum edge matching for reconfigurable computing. In *Reconfigurable Architectures Workshop at 13th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006)*, Rhodes, Greece, April 2006.

[8] Xilinx, Inc. Planahead: www.xilinx.com/planahead.