Enhancing Resource Utilization with Design Alternatives in Runtime Reconfigurable Systems

Alexander Wold, Dirk Koch, Jim Torresen
Department of Informatics, University of Oslo, Norway
Email: {alexawo,koch,jimtoer}@ifi.uio.no

Abstract—Average resource utilization in reconfigurable systems is dependent on how effectively modules are placed. Resources may be left unused because modules usually have different resource requirements, and consequently different module bounding boxes. When fitting such different modules on an FPGA, the placement can be imperfect. In order to investigate the effect of module design alternatives on resource utilization, we model the placement problem as a constraint satisfaction problem. In particular we are interested in real world FPGAs. The constraint satisfaction problem is solved using a constraint solver. The constraint problem is modelled as a minimization problem in order to attain the optimal placement. We have investigated module design alternatives which include representing the modules with different layout. We have found an increase in average resource utilization of 11% by considering module design alternatives. The computation time is short, so the method could be applied as a part of an interactive tool.

Index Terms—Floorplanning, field-programmable gate array (FPGA), design alternatives, reconfigurable architectures, constraint programming.

I. INTRODUCTION

Runtime reconfigurable system design has been a research topic for some time. The benefits of runtime reconfiguration include increased resource utilization and lower energy consumption. However, runtime reconfiguration comes with an associated cost. This cost is measured in both area utilization and reconfiguration time. The overhead incurred when implementing runtime reconfigurable systems needs to be low in order not to offset the benefits offered by such systems. In order to increase area utilization, we will in this paper estimate the impact of module placement when module design alternatives are taken into account in our work.

Typically a module is represented with only one implementation. In this paper, we look at several different layouts of a module providing similar functionality. The term module design alternatives describes functionally equivalent modules with different layouts. These modules have similar performance and functional requirements. Module design alternatives may also differ in the use of dedicated resources. The module, and the module alternatives, do not necessarily consume the same amount of resources. Examples of such design alternatives are shown in Figure 1. Here, a module is represented with five different layouts. In this example, the module and the module alternatives consume the same amount of resources. Run time reconfigurable systems where the module is preempted and the state stored, restoring the module with a different design alternative would present a problem in restoring the state. Consequently, we do not consider changing design alternatives at run-time.

Fragmentation is a measure of resources left unused because modules usually have different resource requirements, and consequently different module bounding boxes. When fitting such different modules on an FPGA, the placement can be imperfect. Module design alternatives are introduced to address the overhead related to fragmentation. The aim is to minimize this overhead by considering module design alternatives in the placement process. Fragmentation will be reduced as a result of considering several design alternatives, as a higher quality fit may be possible.

Another factor influencing average resource utilization is dedicated resources. Embedded memory, multipliers, and other dedicated resources reduce the placement possibilities. Modules placed on a heterogeneous FPGA cannot be placed freely, as the layout of the module and the location of the dedicated resources impose constraints on the placement. Newer FPGAs are increasingly more heterogeneous. This presents placement restrictions which need to be reflected in the placement model. In runtime reconfigurable systems, this limits module placement. [1] shows an average resource utilization of 36% when placing modules in a reconfigurable region on a heterogeneous FPGA. While previous generations of FPGAs also contained dedicated resources, these resources were located regularly aligned in columns. In current generations of FPGAs dedicated resources are no longer located in such regular patterns, but are spread more irregularly over the device. Furthermore, some resource columns differ from their resource type (e.g. they contain clock resources). With a homogeneous FPGA, the placement model does not need to incorporate awareness of the location of the different types of resources. However, when this is not the case, dedicated resources put additional
The placement model incorporates awareness of the amount of reconfigurable modules added and removed. In order for new modules to be placed at run-time, the space needs to be managed. Ahmadinia et al. show in [5] that online placement by managing the occupied space is more efficient than free space management. A graph-based approach to derive optimal placement of rectangular modules is presented in [6]. An exact solution of the 2D bin packing problem is described in [7]. The bin packing problem is approached as a constraint satisfaction problem in [8] where a geometrical constraint solver is presented. These references relate to placing modules in a homogeneous xy-plane. This is less applicable with respect to current FPGAs as the current reconfigurable resources are heterogeneous rather than homogeneous.

The different resources on current FPGAs limit placement possibilities. A heterogeneous model reduces the possible locations where modules can be placed, which negatively affects average resource utilization. In [9] it is proposed to mask out dedicated resources in order to increase placement possibilities. However, the effect of not using dedicated resources is detrimental to performance and area as shown in [2]. Thus, possible positive effects run-time reconfigurable systems have on performance [10] and reduced power [11] could be achieved through optimal module placement in reasonable time.

The majority of publications aim to enhance the resource utilization, which also improves the service level of a system (i.e., the amount of module requests that can be fulfilled). The majority of publications can be classified in multiple categories:

1. Offline versus online
2. Homogeneous versus heterogeneous resource model
3. Rectangular versus universal module shape
4. Free space management versus occupied space management
5. 1D slot-style versus 2D-grid module placement style

Within the above classification, the work presented in this paper relates to offline placement in a heterogeneous environment of modules with universal shape.

In [4], module placement is approached as a fully flexible 2D packing problem. Fragmentation occurs in an online nondeterministic context-switching environment as a result of modules added and removed. In order for new modules to be placed at run-time, the space needs to be managed. Ahmadinia et al. show in [5] that online placement by managing the occupied space is more efficient than free space management. A graph-based approach to derive optimal placement of rectangular modules is presented in [6]. An exact solution of the 2D bin packing problem is described in [7]. The bin packing problem is approached as a constraint satisfaction problem in [8] where a geometrical constraint solver is presented. These references relate to placing modules in a homogeneous xy-plane. This is less applicable with respect to current FPGAs as the current FPGAs are heterogeneous rather than homogeneous.

The different resources on current FPGAs limit placement possibilities. A heterogeneous model reduces the possible locations where modules can be placed, which negatively affects average resource utilization. In [9] it is proposed to mask out dedicated resources in order to increase placement possibilities. However, the effect of not using dedicated resources is detrimental to performance and area as shown in [2]. Thus, possible positive effects run-time reconfigurable systems have on performance [10] and reduced power [11] could be achieved through optimal module placement in reasonable time.
would be limited. In [12], authors present a fine grained 2D-module placement in order to reduce fragmentation. Fine grained tiles reduce internal fragmentation. In order to increase average resource utilization, fine grained tiles depend on high quality placement. Resources may be left unused because of fragmentation.

Hagemeyer et al. [13] presents an integrated design flow for developing run-time reconfigurable embedded systems. Both [14] and [1] present placement models for heterogeneous run-time reconfigurable systems. These placement models take the different resources into account by allocating suitable regions for a set of modules at design time.

III. PLACEMENT MODEL FORMULATION

The placement model presented here comprises the module formulation, the partial region definition and placement constraints. The constraint definitions are applicable to placing modules on a heterogeneous reconfigurable region.

A. Module formulation

A module consists of a set of one or more shapes. A shape consists of one or several tile sets. One or several tiles constitutes a tile set. A module containing several shapes described in such a way can be seen in Figure 1. Here a module is represented with several layouts or shape alternatives. Figure shows the effect of placing modules with shape alternatives. Figure 1 shows optimal placement with design alternatives. In this figure, each module is represented with two layouts. The second layout is a 180 degree rotation of the first layout. The bottom figure shows optimal placement without design alternatives.

The tiles represented in each tile set have an origin coordinate pair and an internal resource type. The resource type represents a physical FPGA resource. The different boxes shown in Figure 3 reflect the resource types. Internal resource types can further be used to represent communication macros for bus attachment.

A tile, $t$, is defined as $t_{x,y,k}$, where $x$ and $y$ are origin coordinates, and $k$ is the resource type. A tile is the smallest unit and has a length and width equal to 1. A tile set, $T$, is defined as a set of tiles, $T_k = \{t_{k1}, t_{k2}, t_{k3}, ..., t_{kn}\}$, where $n > 0$, i.e. the set is not empty. The tiles which constitute a tile set has identical resource type. A shape, $S$, is formally defined as a nonempty set of tile sets, $S = \{T_1, T_2, T_3, ..., T_n\}$, where $n > 0$. A shape represents one possible physical implementation of a module, as shown in 1. The origin coordinates of a tile are relative, thus, a shape can be freely moved in the xy-plane. A module, $M$, is defined as a set of shapes, $M = \{S_1, S_2, S_3, ..., S_n\}$, where $n > 0$.

A module is defined as one or several distinct groups of tiles. These groups of tiles form several tile sets. The tile sets that constitute a shape represent design alternatives, as shown in Figure 3 design alternatives are described by having each module represented with layout variations. For simplicity, in this figure the different module shapes contain a similar number of tiles with associated internal resource properties. However, there is no constraint defined in the placement model which limits the different shapes or layouts, in this way. Therefore, it is possible to provide design alternatives where the shapes does not consist of an equal number of tiles and resource types.

Usually the tiles, which constitute a shape, are located directly adjacent to one another. However, this is not a requirement. Routing restrictions place some limits on the freedom to construct modules with nonadjacent tiles. We therefore do not consider such design alternatives.

B. Partial region formulation

The partial region model consists of tiles and regions in a fashion similar to that of the module formulation. The partial region layout is defined as a set of tiles with different internal resource types. These internal resource types represent different resources such as embedded memory, multipliers, IO, and logic. A partial region model containing tiles of different resources is shown in Figure 3. The different sizes of the boxes represent different resources. Empty boxes are unused resources.

The partial region $P$ is defined as a nonempty set of tile sets: $P = \{T_1, T_2, T_3, ..., T_n\}$, where $n > 0$.

This partial region model is modelled after a real world FPGA. As shown in Figure 3, the different resource types do not normally have the same size, hence the different sizes of the boxes. Embedded memory consumes more area than multipliers and logic. On an FPGA, a tile set is a cluster of identical resources like, for example, a set of block ram (BRAMs) or configurable logic blocks (CLB)s on FPGAs from Xilinx.

A multiplier module is modelled as a tile set $T$ consisting of four tiles. This forms the set $T_k = \{t_{0,0,k}, t_{0,1,k}, t_{1,0,k}, t_{1,1,k}\}$, where $k$ represents the resource type, in this case a multiplier resource. A CLB forms the tile set $T_k = \{t_{0,0,k}\}$ consisting of a single tile element. The $x$ and $y$ coordinates of a tile is absolute when it is an element of a partial region set. This is different to that of the module formulation.

The partial region model encompasses the reconfigurable and the static regions of the device. A bounding box consuming about 50% of the partial region is shown in Figure 4c. This area is allocated for the static region. Usually the static regions are modelled in this way. The rest of the device is dedicated to reconfigurable modules. The static region is implemented as a tile or several tiles with a resource type defined as "not available."

C. Constraining module placement

A set of constraints is imposed on the placement of modules on the partial region. These constraints eliminate
invalid placements. For example, invalid placements include placing modules outside the partial region and placing modules overlapping one another.

Figure 4 shows how placement constraints affect the module placement. In 4a, module placement is constrained within a bounding box consisting of the reconfigurable partition. Modules cannot be placed outside the bounding box. Figure 4b shows how module placement is restricted as a result of different resources. Valid placement of the module is limited to the gray areas. In 4c, the reconfigurable region covers only portions of the total region. Modules cannot be placed in unused portions of the area outside the bounding box. Figure 4d shows a possible placement for a module. Other modules cannot be placed overlapping this area.

We define the complete solution set, \( A \), as the disjoint union of all modules:

\[
A = \bigcup_{i=0}^{I} M_i
\]  

The partial region set is defined in the previous section. In order to distinguish elements in the \( P \) set from elements of the \( A \) set, elements belonging to the \( P \) set are marked such that \( t_{x,y,k} \in A, t_{x,y,k} \notin P \), and similarly \( t'_{x,y,k} \notin A, t'_{x,y,k} \in P \).

Of the complete solution set \( A \), the solutions which are of interest are the subsets where the shapes are placed inside a defined region. This implies that, for all tilesets in a shape, the tiles in the tilesets must be placed inside the constrained region. The constrained region represents the area of the reconfigurable device. Boundaries are defined as constraints. This is shown in Figure 4c.

The set, \( M_{ai} \), is defined as the subset of \( M_i \) for which all tiles \( t \) in all shapes lie within the constrained region:

\[
M_{ai} = \left\{ S \in \bigcup_{i=0}^{I} M_i \iff \forall t \in S, t_x \leq t'_x, t_y \leq t'_y \right\}
\]

Further, each shape consists of a number of tilesets. These tilesets \( T \), consisting of tile elements, have a corresponding resource identifier \( k \). The tilesets are only considered having a valid location when they are placed on a tileset of the partial region with identical resource identifier. Thus, shapes \( S \) for which this constraint is not satisfied are not part of the subset \( M_{bi} \) of \( M_i \). This adds further restrictions on the exact placement of the module, as the shapes cannot be freely placed on a heterogeneous FPGA model, as shown in Figure 4b. The set, \( M_{bi} \), is defined as the subset of \( M_i \) where the tileset resource type corresponds to the resource type of the partial region tileset. This subset is formulated as follows:

\[
M_{bi} = \left\{ S \in \bigcup_{i=0}^{I} M_i \iff \forall T_k \in S, T_k = T'_k \right\}
\]

Valid placement of a single shape is possible with the previously defined constrained unions. In order to allow several shapes to be placed without overlapping each other, a non-overlapping constraint is defined. This constraint prevents the placement of modules as shown in Figure 4d. Two modules may both be placed in this area, but not at the same time. \( S' \) represents a shape in \( M_j \) where \( j \neq i \). If the subset of \( S' \cap S = \emptyset \), then \( S \) and \( S' \) do not overlap. The subset, \( M_{ci} \) of \( M_i \), is defined as follows:

\[
M_{ci} = \left\{ S \in \bigcup_{i=0}^{I} M_i \iff \emptyset = S \cap S' \right\}
\]

From the previously defined subsets, \( M_{ai}, M_{bi}, M_{ci} \), we derive the following set, the intersection of the aforementioned subsets, which satisfies all constraints:

\[
A' = \bigcap_{i=0}^{I} \left\{ M_{ai} \cap M_{bi} \cap M_{ci} \right\}
\]

In other words, a valid solution \( A' \) must ensure that all module shapes are located in the configurable region, that the placement follows the resource types, and that different modules do not overlap.

D. Optimized module placement

The set \( A' \) contains all valid solutions. In this set, the different solutions vary with respect to average resource utilization. In order to obtain the subset with the highest average resource utilization, the problem is approached as an optimization problem. As a consequence of \( M_{ai} \), defined in equation 2, the elements of \( A' \) are constrained in both the \( x \) and \( y \) direction. The set of solutions with minimal height is the optimally placed subset, with the highest average resource utilization. This is defined as \( A'' \) as follows:

\[
A'' = \min \{ A' \}
\]

IV. CONSTRAINT PROGRAMMING

Constraint programming is a method of solving a problem by stating a set of constraints which must be satisfied in the
solution. Using constraint programming techniques a placement problem can be expressed as a set of constraints imposed on possible placement alternatives. Constraint programming is used because of the intuitive nature of expressing placement constraints in order to obtain optimal placement.

The bin packing problem is known as a NP-hard problem. Optimal module placement is a slight variation of the bin packing problem. Different resources reduce the number of possible placement solutions. This leads to decreased computation time. At the same time, the introduction of module design alternatives increases the number of possible solutions. This results in increased computation time. Interactive use requires short computation time. As shown in the previous section, module placement can be described as set constraints. It is therefore interesting to see whether or not the placement of modules can be solved efficiently using a constraint programming framework.

The placement model can be implemented directly from the constraint definitions, module and partial region models defined in the previous section. A branch-and-bound function can be used to prune the search space of solutions which do not lead to solutions satisfying the constraints. The remaining set of solutions, assuming the set is not empty, forms a subset of solutions whose optimal solution can be found. In a number of applications a solution which satisfies the constraints is adequate, and an optimal solution is not required. In order to achieve higher average resource utilization, we are interested in the optimal placement.

The constraints defined in the placement model do not cover average resource utilization. Constraints are limited to defining acceptable placement locations for the modules. In order to address average resource utilization per area, the placement is approached as a minimization problem. This increases computation time.

In order to take advantage of the geometrical properties of the placement problem, we have implemented the placement program based on the geost constraint kernel by N. Beldiceanu et al. [8]. The geost kernel is a geometrical constraint solver handling placement of objects.

In the geost constraint kernel, a module is defined as a finite set of shapes. A shape is defined as a set of boxes. A box is defined with an origin and a size. A box is similar to a tile set as defined in section III.

The Geost kernel implements placement constraints. These constraints are similar to the constraints formulated in the previous section.

The geost kernel is limited in that it originally only considered the geometrical properties of a shape. Without a constraint definition covering different resource types, a heterogeneous placement model cannot be implemented. In order to solve this, the following improvements have been implemented:

1) the geost definition of a box is extended with a resource property.
2) the geost kernel implements a constraint defining regions where modules are not placed. This is constraint is extended with a resource property. With the implemented extensions, the constraint can be used to model FPGA resources.

With these improvements, it is possible to achieve optimal module placement within a constraint programming framework.

<table>
<thead>
<tr>
<th>Type</th>
<th>Mean Area Util.</th>
<th>Mean Time</th>
<th>CLB</th>
<th>BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>No design alternatives</td>
<td>53%</td>
<td>2.55s</td>
<td>20-100</td>
<td>0-4</td>
</tr>
<tr>
<td>Design alternatives</td>
<td>65%</td>
<td>10.82s</td>
<td>20-100</td>
<td>0-4</td>
</tr>
<tr>
<td>Change</td>
<td>22%</td>
<td>-8.26s</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table I

IMPACT OF MODULE DESIGN ALTERNATIVES ON AREA UTILIZATION AND EXECUTION TIME

V. RESULTS

In order to evaluate the impact of module design alternatives with respect to average resource utilization, we place modules with and without design alternatives on a heterogeneous FPGA model. Modules are placed based on the model and the implementation introduced in the previous sections. The effect of design alternatives on the execution time of the placer is also studied. The results are summarized in Table I.

A. Module placement

The test results are derived from 50 runs of placing 30 automatically generating modules with shapes similar to that shown in Figure 1. The module resource requirements of the generated modules varies between 20 and 100 CLBs, and between 0 and 4 embedded memory blocks. The module alternatives considered include variants in which the module is rotated 180 degrees and additionally have different internal and external layout. Internal layout refers to a module which has the same bounding box, but for which dedicated resources have different positions within the module. External layout refers to a module with different bounding box. A module is represented with four different module shapes. When placing modules with design alternatives, 30 modules yield 120 different shapes.

Dedicated resources such as embedded memory have rectangular rather than square layout. Modules using embedded memory having the external bounding box rotated 90 and 270 degrees can therefore not be placed without changes to the internal position of resources.

The effect of design alternatives is an increase in resource utilization from 53% to 65%. The improvement in resource utilization is the result of the increased number of placement possibilities. Naturally, this is dependent on the shapes of the modules.

B. Execution time

While improvement is shown with respect to average resource utilization, the additional number of shapes considered when placing modules with design alternatives, has negative impact on the total execution time of the placer. The average execution time increases from an average of 2.55 seconds to an average of 10.82 seconds on a 2.8GHz desktop workstation when considering four design alternatives. This is shown in Table I. The execution time limits the method to offline placement computation. The total number of shapes when placing modules with design alternatives increases from 30 to 120.

VI. CONCLUSION

The main contribution in this paper is a placement model based on constraint programming incorporating module design alternatives. We have quantified the effect of module design
alternatives on average resource utilization and execution time of the placing algorithm.

The module placer is implemented using constraint programming techniques. It is based on a previously published constraint solver ([8]), extended according to our model to consider a heterogeneous FPGA with different resources. The constraint problem is modelled as a minimization problem in order to derive the optimal placement.

Our results show that by introducing design alternatives, we have increased the average resource utilization from 53% to 65%, an increase of 11%. The improvement is the result of decreased fragmentation. Average resource utilization in reconfigurable systems reflects how effectively modules are placed. The increase in average resource utilization is dependent on the layout of the modules and module design alternatives.

A key goal in this work is to achieve acceptable placement in a reasonable time for real world FPGAs. Reasonable time in this respect refers to execution time acceptable for interactive use. The introduction of module design alternatives increase the execution time of the placer. The experiments show a negative impact, with an increase in execution time from about 2.55 seconds to about 10.82 seconds. This is still acceptable when used for development.

ACKNOWLEDGMENT

We are supported by the Norwegian Research Council under grant 191156/V30.

REFERENCES


