Module Allocation for Dynamically Reconfigurable Systems

Xue-jie Zhang and Kam-wing Ng

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N. T., Hong Kong
{xjzhang, kwng}@cse.cuhk.edu.hk

Abstract. The synthesis of dynamically reconfigurable systems poses some new challenges for high-level synthesis tools. In this paper, we deal with the task of module allocation as this step has a direct influence on the performance of the dynamically reconfigurable design. We propose a *configuration bundling* driven module allocation technique that can be used for component clustering. The basic idea is to group configurable logic together properly so that a given configuration can do as much work as possible, allowing a greater portion of the task to be completed between reconfigurations. Our synthesis methodology addresses the issues of minimizing reconfiguration overhead by maintaining a global view of the resource requirements at all times during the high-level synthesis process.

1 Introduction

A dynamically reconfigurable system allows hardware reconfiguration while part of the reconfigurable hardware is busy computing, and allows a large system to be squeezed into a relatively small amount of physical hardware[1]. Though very promising, the development of dynamically reconfigurable systems faces many problems.

Since the configuration changes over time, one major problem is that there needs to be some way to ensure that the system behaves properly for all possible execution sequences. For this time-multiplexed reconfiguration to be realized, a new temporal partitioning step needs to be added to the traditional design flow. Some researchers have addressed temporal partitioning heuristically, by extending existing scheduling and clustering techniques of high-level synthesis[2][3][4]. In an earlier work[5], we presented a design model for abstracting, analyzing and synthesizing reconfiguration at the operations level.

In addition to making sure that a temporal partitioning be done correctly and producing a functionally correct implementation of the desired behavior, another important problem is how to produce the best implementation of functionality. With normal FPGA-based systems, one wants to map the configurable logic spatially so that it occupies the smallest area, and produces results as quickly as possible. In a dynamically reconfigurable system one must also consider the time

to reconfigure the system, and how this affects the performance of the system. Configuration can take a significant amount of time, and thus reconfiguration should be kept to a minimum. This is in general a challenging problem to address, with almost no current solution[6].

In this paper, we present an efficient high-level synthesis technique which can be used to synthesize and optimize dynamically reconfigurable designs. In particular, we concentrate our investigation on the task of module allocation. Dynamic reconfiguration extends the module allocation space by an additional dimension. The optimizing criteria in dynamic resource allocation also shift from a single static netlist to several configurations of the design. We must account not only for temporal partitioning and scheduling effects but global considerations as well, such as the resource requirements of all configurations, reconfiguration overhead, and the combination of all of the above. We have addressed these issues by using a *configuration bundling* technique that balances the advantages of dynamic reconfiguration against the added cost of configuration time by maintaining a global view of the resource requirements of all temporal partitions at all times during high-level synthesis.

2 Problem Formulation

The contribution of this paper can be seen in the context of our previous work on a design model[5]. Our approach uses an extended control/data flow graph (ECDFG) as the intermediate representation of a design. The CDFG is extended by abstracting the temporal nature of a system in terms of the sensitization of paths in the dataflow. An ECDFG is a behavioral-level model. An ECDFG representation of system behavior consists of three major parts: (1) possible execution paths which are described by the product of the corresponding guard variables, (2)temporal templates which lock several configuration compatible operations into temporal segments of relative schedules, (3) a control and data flow graph (CDFG) describing data-dependency or control-dependency between the operations. Interested readers are referred to the original references for the details about ECDFG.

In high-level synthesis, module allocation is an important task which determines the number and types of RTL components to be used in the design. Since we have encoded the temporal nature of synthesizing such systems by *temporal templates*[5], the module allocation process may be translated into a two-dimensional placement problem of temporal templates. Instead of considering individual CDFG nodes, we restate the dynamic module allocation problem in terms of temporal templates, a given spatial and temporal placement of configurable logic resources used by some temporal templates for a range of time constraints represents a possible configuration. The module allocation problem for dynamically reconfigurable logic involves not only generating the configuration for each of the temporal templates, but also reducing the reconfiguration overhead incurred. Our problem can be formally defined as follows:

Problem 1. Let $F = \{F_1, F_2, ..., F_m\}$ be a set of function units which can be implemented on reconfigurable logic, and $C = \{C_1, C_2, ..., C_n\}$ be a set of possible configurations of the configurable logic units. Given an extended CDFG (ECDFG) $G = (V, E, \xi, \zeta)$ with a set of temporal templates in a given order $TT = (TT_1, TT_2, ..., TT_p)$, where $TT_i \in F$, find an optimal sequence of configurations $CS = (CS_1, CS_2, .., CS_q)$ for temporal template TT, where $CS_i \in C$ which minimizes the reconfiguration cost R. R is defined as

$$R = \sum_{i=2}^{q} \delta_i \tag{1}$$

Where δ_i is the reconfiguration cost in changing configuration from CS_{i-1} to CS_i.

In the remaining sections, we use a new *configuration bundling driven technique* to address the module allocation problem.

3 Configuration Bundling

The basic idea is to group logic together properly so that a given configuration can do as much work as possible, allowing a greater portion of the task to be completed between reconfigurations.

We illustrate our concept with the help of a motivating example. Consider three temporal templates of an extended CDFG shown in Figure 1. Furthermore, assume that all operations finish in a single cycle and that all temporal templates have to be implemented in three clock cycles. If each temporal template is allocated as a single configuration, the first temporal template (shown in Figure 1(a)) requires a module allocation of *five functional units* namely $\{3\ adders, 1\ multiplier, 1\ subtractor\}$. Similarly, the second and third temporal templates (shown in Figure 1(b)-(c)) can be implemented with module allocations of $\{\ 2\ adders,\ 2\ multipliers,\ 2\ subtractors\}$ and $\{1\ adder, 1\ multiplier, 3\ subtractors\}$ respectively.

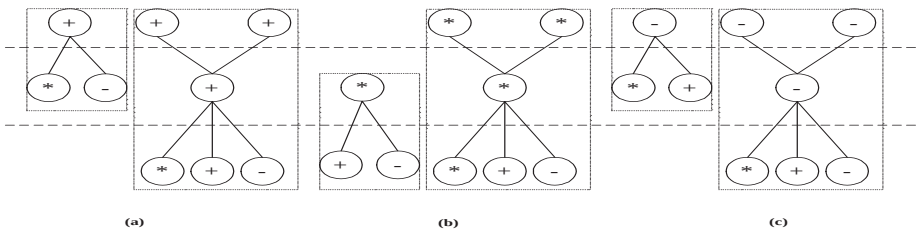

Fig. 1. A Motivating Example

A straightforward approach to optimize the module allocation of the three temporal templates as a dynamically reconfigurable design involves considering the granularity of the reconfiguration. Resource requirements of the temporal templates can be reduced significantly by maintaining a global view of the resource requirements of all temporal templates at all times during the synthesis process. In fact, the three temporal templates can be implemented using a configuration granularity of *two adders, two multipliers and two subtractors.* In this research, we have developed a configuration bundling technique to reduce the reconfiguration overhead. The concept of *configuration bundling* can be defined as follows:

Definition 1. *Given an extended CDFG (ECDFG)* $G = (V, E, \xi, \zeta)$ *with a set of temporal templates* $TT = \{TT_1, TT_2, ..., TT_n\}$*, a configuration bundle is a subset of* TT *such that the hardware resource requirements of individual temporal template in this subset* $\{TT_{i_1}, TT_{i_2}, ..., TT_{i_m}\}$ *can be implemented by an overall resource allocation schema.*

Configuration bundling is a synthesis technique where n temporal templates are bundled into at most m groups so that each temporal template belongs to at least one bundle and the objective function is optimized. Following configuration bundling, each bundle is synthesized into a separate configuration. The basic idea behind our configuration bundling technique is to attempt to identify and bundle temporal templates with similar computation topology and hardware types into compatible groups, such that these groups may be used to determine the choice of granularity for configurations that optimize the reconfiguration overhead. In particular, the following compatibility issue should be considered during the configuration bundling process.

3.1 Bundling Compatibility of Temporal Templates

If two temporal templates with disparate topologies are implemented in temporally consecutive configurations the attendant configuration overhead will be significant. In the worst case, each functional unit has to be reconfigured and this increases the time of reconfiguration. Therefore, topological similarity between temporal templates should be considered for bundling into the same group. For example, in Figure 2 *Temporal Template 2* can be bundled into a configuration implementing temporal template 4 with almost no reconfiguration overhead.

In addition, resource compatibility is an important issue during *configuration bundling.* For example, in Figure 2, while *Temporal Templates* 2, 3 and 4 use subtractors and multipliers, *Temporal Template* 1 uses adders. Therefore, bundling *Temporal Template* 1 with either *Temporal Template* 2 or 3 or 4 does not yield justifiable benefit for reducing the reconfiguration overhead. On the other hand, based on the compatibility of the functional unit types, *Temporal Templates* 2, 3, 4 are good candidates to be bundled into the same group.

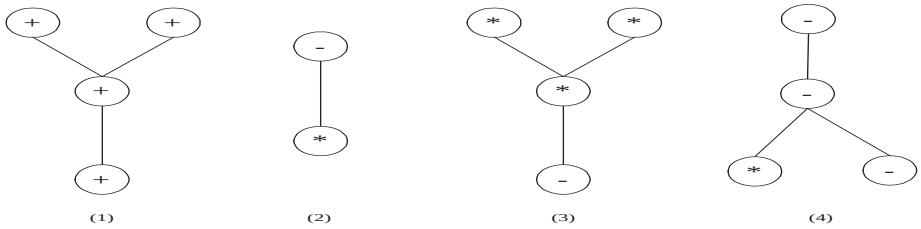

Fig. 2. Compatibility of Temporal Template

3.2 Measure of Configuration Bundling

Configuration bundling should take into account trade-offs between maximizing static resource requirements and minimizing reconfiguration overhead in space. Therefore, a configuration bundle will have the smallest area and the scope for maximum resource usage if the temporal templates in a bundle are compatible with one another. Based on the above observations we have developed a measure to identify bundling compatibility between temporal templates. We first outline the parameters of the function for bundling below.

- B: Set of bundles $B_1, B_2, ..., B_k$ for a given TT that describes a set of possible configuration bundling.
- $N_{F_j}(TT_i)$: the number of configuration of functional unit F_j for temporal template TT_i.
- $Area_{F_i}$: the area of a configuration of a functional unit F_i

Given a temporal template TT_i, the following is an estimate of the area of the temporal templates:

$$Area_{tt_i} = \sum_{f \in F} N_F(TT_i) \cdot Area_f \tag{2}$$

If a bundle B_i has n_i temporal templates, then the area of the bundle is estimated as below.

$$Area_{B_i} = \sum_{f \in F} max_{tt \in B_i} N_f(tt) \cdot Area_f \tag{3}$$

The larger the difference between these areas of temporal templates, the more incompatible the temporal template will be with the remaining temporal templates in the bundle. Given a temporal template TT_j for consideration for bundling in B_i, the incompatibility can be obtained as the following and is used to weigh the candidate solutions.

$$\delta_{B_i,TT_j} = \sum_{f \in F} |max_{TT \in B_i} N_f(TT) - N_F(TT_j)| \cdot Area_f \tag{4}$$

4 Configuration Bundling Driven Module Allocation Algorithm

Since there are several temporal templates in a range of time and module allocation, simultaneously considering all the temporal templates and their respective constraints is difficult. We propose to allocate the hardware resources from a range of time by considering one temporal template at a time. In particular, the following three issues must be taken into account:

- the allocated hardware resource due to the previously considered temporal templates
- the estimated hardware resource of the remaining temporal templates
- the hardware resource required by the candidate temporal templates

Here, temporal templates are first bundled randomly. Then, a source configuration bundle is randomly chosen. From such a configuration bundle, an incompatible temporal template is selected and moved to another configuration bundle where the temporal templates are compatible with the selected temporal template. The hardware area of all configurations is then computed, and the current bundling configuration is saved if it is the best so far. The process continues until no more improvement is obtained for a given number of iterations.

For each configuration bundle $B_i \in B$, the module allocation algorithm is outlined below. An initial module allocation A_{B_i} for each configuration bundle B_i is first derived. Starting with a temporal template with the most resource requirements, a feasible module allocation for the entire bundle is obtained. From the total resource allocated to the configuration bundle A_{B_i}, the module allocation R_{TT_i} for each candidate temporal template $TT_i \in B_i$ is obtained. Then, allocation and scheduling of the design are carried out using this module allocation technique.

4.1 Initial Module Allocation

Let N_{ij} be the maximum bound on the necessary amount of resource of a certain configuration type C_j of functional unit for the temporal template TT_i of a configuration bundle. For each resource type C_j and for each *temporal template* TT_i of a configuration bundle B_i, relaxation based scheduling techniques are used to derive an estimate of N_{ij}. For a configuration bundle B_i, a global minimum bound of resource requirements N_j is used as the initial allocation for the configurable logic C_j.

$$N_j = max_{TT_i \in B_i}(N_{ij}) \tag{5}$$

This is based on the fact that there will be at least one *temporal template* in the *configuration bundle* that requires at least these many hardware configurations of type C_j.

4.2 Ordering and Allocating Temporal Templates

Within our methodology, the ordering of temporal templates in the same configuration bundle has an impact on resource usage and reconfiguration overhead of the resulting resource allocation. A good order for module allocation of temporal templates is important because this order has a pronounced impact on the final resources allocation and the overall performance of the system. The proposed algorithm for ordering temporal templates include two stages called clustering and scheduling. The objective of the algorithm is to group temporal templates such that they may subsequently be allocated and scheduled.

When considering functional locality in the module allocation process, it is better to schedule and allocate together temporal templates contributing to the same *join node* in the ECDFG, because this could help in the scheduling and allocation of relative temporal templates at higher levels. Therefore, clustering temporal templates is the first step in the temporal templates ordering process. The *cones partitioning algorithm* provides the basis for our clustering stage[8][9].

Once temporal templates are partitioned into clusters, the cluster-based list scheduling and allocation algorithm orders the temporal templates in the same configuration bundle. Our algorithm combines scheduling with module allocation into subsequent configurations for temporal templates in the same configuration bundle, while considering functional locality of the configuration bundle. There are two main steps in our list scheduling algorithms: the formation of clusters and list scheduling temporal templates.

5 Experimental Results

In this section we present results to illustrate the effectiveness of the configuration binding technique. In order to experimentally verify the concept of configuration bundling driven module allocation, we used three popular high-level benchmarks - elliptical wave filter (EWF), finite impulse response filter (FIR) and bandpass filter (BF) - for optimizing the overall resource allocation as well as the reconfiguration overhead. We assume the following configurations for addition and multiplication operations: look-ahead adder (Area = 1, latency = 1) and a two-stage multiplier (Area = 4, latency = 2). Figure 3 shows the component requirement for the static and configuration bundling driven module allocation.

Bundles	Static module allocation (Area)	Bundling driven module allocation (Area)	Reduction
{EWF,FIR,BF}	24	11	54.2%
{EWF,FIR}, {BF}	24	17	29.2%
{EWF},{FIR,BF}	24	13	45.8%

Fig. 3. Bundling to minimize reconfiguration cost

We have also combined our front-end algorithms with the existing DRL scheduling algorithms[2] back end for demonstrating our results. DRL scheduling algorithms do not consider the module allocation problem. We compare results of the combined algorithms with a single DRL approach[2] as shown in Figure 4, where t_e, n_p, n_f and λ represent the total data-path execution time, the number of partial and the number of full reconfigurations and the graph latency respectively.

Benchmarks	Total area	Combined approach				DRL			
		t_e	n_p	n_f	λ	t_e	n_p	n_f	λ
Elliptic wave_filter	15	15	25	0	15	17	1	2	17
	10	15	25	0	15	17	4	2	17
	6	15	24	1	15	17	2	8	17
	15	16	12	0	16	17	5	0	17
	10	18	13	1	17	18	9	0	18
	6	20	17	0	20	24	19	0	24
	15	19	3	0	19	18	8	0	18
	10	26	9	0	26	21	16	0	21
	6	28	12	0	28	37	1	9	19

Fig. 4. Synthesis result and comparison

The results in Fig.4 have shown that the use of the combined algorithm will lead to a faster execution time compared with a single DRL scheduling implementation, and with considerably smaller area. When the DRL scheduling is used alone, more control steps result but when scheduling together with our module allocation is performed the partial reconfigurations will frequently occur instead of the full reconfigurations. This is expected as our algorithm aims at producing a short reconfiguration time by maintaining a global view of the resource requirements of all temporal templates at all times during the synthesis process.

6 Conclusions and Acknowledgments

We have presented a new module allocation technique in this paper. It is based on a configuration bundling heuristic that tries to allocate configurable logic resources by maintaining a global view of the resource requirements of all temporal templates. The most important value of the configuration bundling driven module allocation technique is that enable trade-offs between the granularity of the configuration and reconfiguration overhead during high-level synthesis process.

The work described in this paper was partially supported by two grants: the Research Grant Council of the Hong Kong Special Administrative Region (RGC Research Grant Direct Allocation - Project ID: 2050196), and Yunnan Province Young Scholar Grant.

References

1. Lysaght and J. Dunlop: Dynamic reconfiguration of FPGAs, More FPGAs, UK:Abingdon EE and CS Books (1994), pp82-94, 1994.
2. M.Vasilko and D.Ait-Boudaoud: Architectural Synthesis Techniques for Dynamically Reconfigurable Logic, Field-Programmable Logic, Lecture Notes in Computer Science 1142, pp290-296
3. J. Spillane and H.Owen: Temporal Partitioning for Partially Reconfigurable Field Programmable Gate, Proceedings of Reconfigurable Architectures Workshop(RAW'98), 1998.
4. M. Kaul and R. Vemuri: Optimal Temporal Partitioning and Synthesis for Reconfigurable Architectures, Proceedings of Design and Test in Europe(DATE'98), 1998.
5. Kam-wing Ng, Xue-jie Zhang, and Gilbert H. Young: Design Representation for Dynamically Reconfigurable Systems, Proceedings of the 5th Annual Australasian Conference on Parallel And Real-Time Systems(PART'98), pp14-23, Adelaide, Australia, September 1998.
6. Scott Hauck and Anant Agarwal: Software Technologies for Reconfigurable Systems, Northwestern University, Dept. of ECE Technical Report, 1996.
7. Ivan Radivojevic and Forrest Brewer: A New Symbolic Technique for Control-Dependent Scheduling, IEEE Trans. on Computer-Aided Design of Integerated Circuit and Systems, vol.15, no.1, pp45-56, Jan. 1996 .
8. D. Brasen, J.P. Hiol and G. Saucier: Finding Best Cones From Random Clusters for FPGA Package Partitioning, IFIP International Conference on VLSI, pp 799-804, Aug. 1995.
9. Sriam Govindarajan and Ranga Vemuri: Cone-Based Clustering Heuristic for List-Scheduling Algorithms. Proceedings of the European Design and Test Conference, Paris, France, March 1997.