A Stochastic Multi-Objective Algorithm for the Design of High Performance Reconfigurable Architectures

Wing On Fung[1], Tughrul Arslan[1, 2]

School of Electronics and Engineering[1]
University of Edinburgh, King's Buildings,
Mayfield Road, Edinburgh, EH9 3JL, UK

Institute for System Level Integration[2]
The Alba Centre, Alba Campus
Livingston, EH54 7EG, UK

wing.fung@ed.ac.uk, tughrul.arslan@ed.ac.uk

Abstract

The increasing demand for FPGAs and reconfigurable hardware targeting high performance low power applications has lead to an increasing requirement for new high performance reconfigurable embedded FPGA cores. This paper presents a multi-objective population based algorithm which given a library of basic blocks and a list of constraints, identifies an optimum reconfigurable embedded reconfigurable core suitable for the target application.

1. Introduction

Modulo placements in regular architectures such as FPGAs and other emerging reconfigurable arrays have been studied extensively, and given the rise in the use of these programmable arrays, they have become increasingly important. Not only do they require performance in terms of timing and delay, the demand for endurable portable electronics implies that minimising routing resources and hence reducing power consumption also plays a significant role in circuit design today.

This paper aims to investigate the use of a population-based heuristic in an attempt to solve the placement problem. Its approach is characterised by its robustness in solving multi-objective and difficult problems. Where a specific heuristic may only be suitable for solving selected problems, the algorithm is useful if the problem space is relatively unknown or large.

2. Background and Overview

The use of heuristics such as simulated annealing [2] to search for optimal placements has been intensely studied. The goals of these heuristics can be categorised into three main areas: timing driven, congestion/routability driven and wirelength driven. There are in general two types of timing-driven algorithms, path-driven and net-driven timing placement.

There are currently commercial [5] and academic [4] placement tools available which are driven by various algorithms. VPR [3] uses bounding box to estimate the total wirelength, and uses path-driven timing analysis with simulated annealing to generate its placement. PATH [4] is an enhancement to VPR. It uses a different net weighting algorithm to VPR, and is implemented by accurately counting the number of paths and assigning the weight of the net according to the number of paths the net is in. The performances of the above tools were tested on T-PEKO [6]. It creates circuit like hierarchies with which their optimal delays are known. Those hierarchies can then be used to test how the placement tools perform. It was found that the delay produced by the current tools are between 10% to 18% away from the optimal critical path delay, hence a significant improvement can still be made from the current tools available.

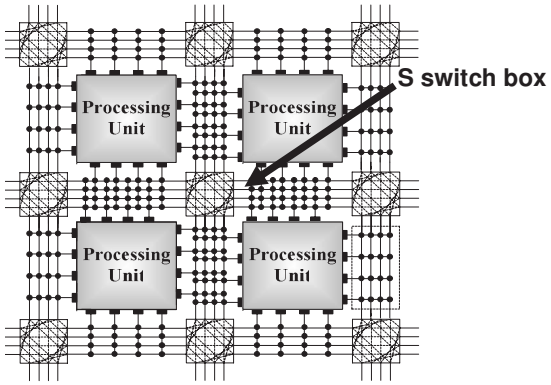


Figure 1. array architecture

1-4244-0054-6/06/$20.00

3. The Models

Nearly 90% of the total power consumed by FPGAs is due to power dissipation through routing resources. With the increasing popularity of reconfigurable hardware being employed in mobile electronics, this gives the motivation to seriously consider minimising power consumption as a primary objective. Accordingly, the model being used should reflect the task of minimising routing resources while the resulting placement should maintain a good delay value. The placement architecture is assumed to be an island-style array with switch boxes and connection boxes switches connecting the nets (Figure 1).

The system is shown in Figure 2. It inputs a netlist and descriptions of the array architecture. Then an evolution [1] based algorithm is implemented to find the likely group of critical paths which will be described below. The following gives the cost factors which model parts of the physical structure of an array. They are used to guide the algorithm to find an optimal solution.

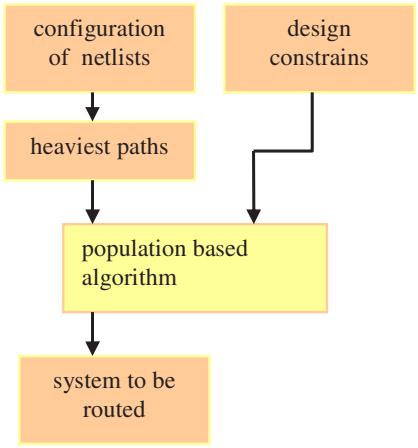


Figure 2. system overview

3.1 Wirelength Cost

The total wirelength of the nets reflects the delay and power dissipation of the circuit and forms the main part of the cost function. The wirelength of a net in a placed circuit is estimated by the Manhattan distance between its output pin to the rest of its input pins. If *inet* is the number of nets in the array and $fanout_i$ is the number of fan-outs of net i, the wirelength cost of a placement would be,

$$C_{wl} = \sum_{i=1}^{inet} \sum_{j=2}^{fanout} \left| diff_{pos}(net_i.block(1), net_i.block(j)) \right|. \quad (1)$$

3.2 Switch Boxes Cost

If the number of active switches used in the circuit is reduced, power dissipation would also be reduced. When the input pin of a net is laying in the same axis as its output pin, it is assumed that they can be directly routed without being diverted by any "S" switch boxes (Figure 1). If a pin is not on the same axis as the output pin, the wire connection between them must go through a "S" switch box (Figure 3). The net is then assigned a weight proportional to the number of pins that are not in alignment.

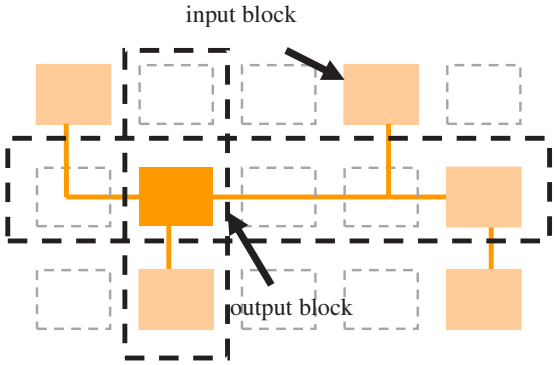


Figure 3. 3 input blocks not in alignment with output

The switches cost for a placement can be found as

$$C_{switch} = \sum_{i=1}^{inet} \sum_{j=2}^{fanout} \begin{cases} 1 & ,(x_{posj} \neq x_{pos1}) \cup (y_{posj} \neq y_{pos1}) \\ 0 & ,otherwise \end{cases}, \quad (2)$$

where x_{posj} and x_{posy} are the coordinates of the fan out pins.

3.3 Finding the Critical Path

The algorithm uses a form of net-driven timing criteria to implement timing analysis on the model. There is likely to be more than one critical path in net-driven timing placement, therefore instead of finding a single critical path, a group of likely critical paths are searched. The resulting population of paths will be used as a cost function in the main heuristic.

If *pop* is the size of the population, P_i is path i of the population, the total cost of the population would be

$$C_{net} = \sum_{i=1}^{pop} length(P_i). \quad (3)$$

The algorithm first generates a population of random paths. The population is then put through a process similar to that of Figure 5, which is described in the next section. A population of long paths from net counting is then determined and is used as part of the final fitness function.

4. The Multi-Objective Algorithm

The costs described above are then input into the heuristic. The following gives a description of the method used to generate an array. The coding of a solution is described in Figure 4. Each logic element of the array is assigned a reference number. The number points to a position in the array for which the logic element occupies.

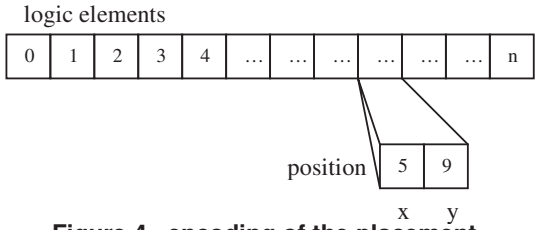


Figure 4. encoding of the placement

The fitness: the objective is to minimise the length of wires and switches usage, hence reducing both power and delays of asynchronous circuits. The fitness function is a combination of the above, and is calculated as follows.

If *ipath* is the number of critical paths generated, the cost function for an individual placement would be,

$$C_{path} = \sum_{i=1}^{ipath} length(P_i).$$

(4)

From (1), (2) and (4), the fitness can then be defined as,

$$fitness = C_i \times C_{WL} + C_j \times C_{switch} + C_k \times C_{path},$$

(5)

where C_i, C_i and C_k are constants that can be varied by the user to tune the algorithm. When the fitness of every individual of the initial population is evaluated, the following operators are used to improve the overall quality of the placements.

Operator 1: randomly selects four placements from the population, and picks the better two according to their fitness; *Operator 2*: combines logic element placements from one selected placement to another in an alternating fashion to create a new placement. It selects a first position in the array and fills it with an element from one individual in the same position. Then it fills the next position with element from the other individual in that position. This is then repeated alternatively, if the element selected is already placed, then an empty or a randomly selected element is placed; *Operator 3*: picks two random positions in a placement and swap the blocks in the positions.

The algorithm is described in Figure 5. Operator 2 passes on useful information from one placement to the next, while operator 3 introduces random changes to the population. After a number of these heuristic operations, a new population is produced. The amount of operations performed is determined by their rates, which are defined by the user. The rates control how well the algorithm will perform and must be fine-tuned if the algorithm is to find a good solution in a reasonable time. The operations are repeated until a satisfied solution is found.

```
initialise population();
best_cost = get_cost(population, Cswitch, Cwl, Cpath);

while placement does not reach criteria:

        for p=0:(population size/2);

            ind_1, ind_2 = op1(p, cost);
            op2(cross_probability, ind_1, ind_2);
            op3(mutate_probability,ind_1);
            op3(mutate_probability,ind_2);
            store(individual_1&2);


        end;

        new_population(n);
        cost = get_cost(population, Cswitch, Cwl, Cpath);
        if (cost >best_cost)
                best_cost=cost;
end;
```

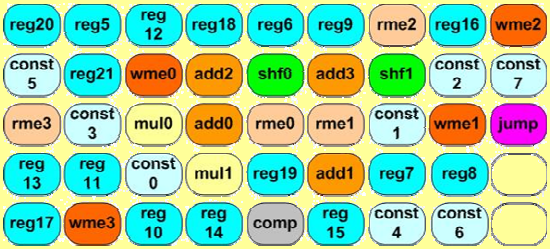
Figure 5. pseudo code of the main heuristic

5. Target Array

The heuristic is tested on two reconfigurable arrays targeting DCT [7] and speech coding separately. The initial architecture targets 27 stages of 32-bit one and two dimensional DCT. This implementation uses 41 logic elements. A placement on a 9x5 array is generated from the input of the 27 configuration descriptions of the array. The placement is then routed using VPR and the results are compared with the designer layout.

The other architecture targets the last loop branch in the Algebraic Codebook Search. It is implemented using 43 logic elements and is shown in Figure 6. A placement on a 9x5 array is generated from 27 configuration descriptions, and it is tested in the same way.

Figure 6. resulted array for speech coding



6. Experimental Result

The algorithm is first fine-tuned to find the best combination of operator rates. Then it was run with a population of 30 to implement both arrays. The small population allows the programme to find a good solution in a reasonable time. It was stop at 3000 iterations when no significant improvement of the placements can be made. Each configuration is then separately evaluated and routed in VPR to get their bounding box and wirelength values.

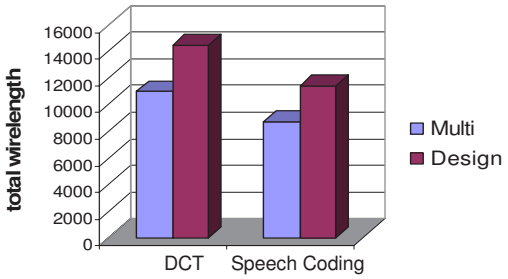
The results of the generated placements for DCT and Algebraic Codebook Search are shown in Figure 7. It compares the total wirelength of the generated array with designer's architecture. In the case of 2-D DCT, there is an overall average of 24% saving in total wirelength. The bounding box is also reduced by a similar rate by 24%. The array for the Codebook Search has an overall bounding box saving of 20%. The total wirelength is also 23% less. Both of the generated architectures have demonstrated improved performance compare with ones placed by designer.

For larger commercial arrays, the program is tested against five circuits of the MCNC LGSynth93 benchmark [8] for FPGAs. Figure 8 shows the critical delays of the placements of both methods.

Although the heuristic succeeded in reducing the initial placement cost by a factor of a third, but comparing with performance of a high performance software tool like VPR, apart from the delay cost of the alu4 array, the heuristic has been outperformed.

There are fundamental constrains which limits the performance of the algorithm implementing in placement architecture. Firstly in every loop of evaluating the population, the algorithm takes up a lot more time than other heuristic algorithms since it requires accessing a large part of memory. Further more, when VPR evaluates a singe placement, it uses information from previous placements to help it calculates the new placement's cost. Where as for the program described, it has to calculate the fitness from scratch. To calculate the wirelength of a circuit with a few thousands nets for every placement is a demanding task. Future research is aimed to optimize the performance of the algorithm as well as producing more accurate model of the reconfigurable arrays.

Figure 7. bounding box cost of DCT/Speech



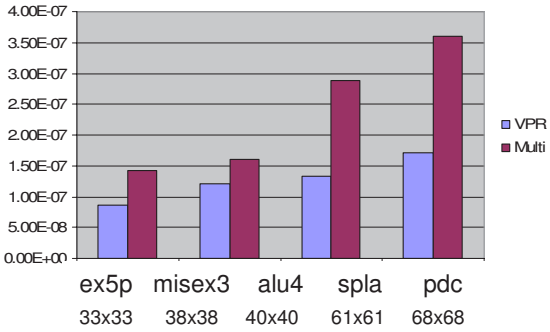


Figure 8. critical delay of MCNC LGSynth93

7. Conclusion

A population based multi-objective algorithm has been presented which aims to produce an embedded reconfigurable core give a library of primitive blocks. The algorithm utilises three cost functions and a set of heuristics in order to identify an optimal embedded reconfigurable architecture for a target specification. We have demonstrated the efficiency of the algorithm by producing embedded reconfigurable cores targeting DCT and motion estimation applications and have compared these with specially tailored efficient designs available in the literature.

References

[1] David E. Goldberg, Genetic Algorithms in Search, Optimization & Machine Learning.

[2] C.-C. Chang, J. Cong, Z. Pan, and X. Yuan. Physical hierarchy generation with routing congestion control. In Proc. Int. Symp. on Physical Design, pages 36.41, 2002.

[3] V. Betz and J. Rose; VPR: A New Packing, Placement and Routing Tool for FPGA Research, Seventh International Workshop on Field-Programmable Logic and Applications, London, UK, 1997, pp. 213 - 222.

[4] Kong, T.; A novel net weighting algorithm for timing-driven placement, Computer Aided Design, 2002 ICCAD 2002, 10-14 Nov. 2002, Pages:172 – 176.

[5] W. Swartz and C. Sechen. Timing-driven placement for large standard cell circuits. In Proc. ACM/IEEE Design Automation Conference, pages 211-215, 1995.

[6] Chin-Chih Chang; Cong, J.; Romesis, M.; Min Xie; Optimality and scalability study of existing placement algorithms Computer-Aided Design of Integrated Circuits and Systems, April 2004 Page(s):537 - 549

[7] S. Khawam, T. Arslan, F. Westall, Synthesizable reconfigurable array targeting distributed arithmetic for system-on-chip applications, 2004 Parallel and Distributed Processing Symposium, (RAW'04)

[8] LGSynth93 MCNC Benchmarks. Obtained from http://www.eecg.toronto.edu/~lemieux/sega/ccts_blif.tar