

Chapter 4

Analysis of Fine-grained Approaches

4.1 Fine-grained Island-style Placement

Before exploring the benefits and drawbacks of macro-based placement for FPGAs, an experimental evaluation is performed of existing placement approaches for FPGAs that consider designs as collections of fine-grained logic blocks rather than as structured components. These fine-grained approaches were first developed for mask programmable gate arrays (MPGAs) [51] and then applied to FPGAs as these devices became available. In general, fine-grained approaches achieve good placement results, but at the cost of a long run time due to the sheer quantity of logic block permutations that must be considered. The goal for this chapter is to quantify the tradeoffs between run time and quality for these approaches so that they may be compared with floorplanning approaches in the next chapter. It is shown through experimental results that required placement time to achieve a specific placement quality level for designs grows at a non-linear rate as design sizes increase, motivating the use of macro-blocks in placement.

The relative lack of routing hierarchy in island-style FPGAs requires placement to be formulated as a global optimization problem rather than as a series of smaller localized subproblems the results of which can later be combined together to form a complete layout. Since routing resources in island-style architectures are organized primarily as a flat mesh with limited segment-to-segment connectivity, local placement optimization that fails to take connections to other parts of the device into account generally leads to results that are far from optimal in terms of the number of routing tracks needed to complete routing successfully. This issue of local versus global placement optimization is revisited in the next chapter when the issue of pre-placed macro-blocks is addressed.

Numerous cell placement algorithms for gate array design styles have been developed and implemented. These approaches can be categorized as either constructive or iterative. Constructive placement [52] typically involves subdividing the total placement problem into smaller pieces through the use of clustering or recursive bipartitioning and then merging intermediate results together in a deterministic

fashion to form a feasible¹ placement. This division of work allows constructive algorithms to converge quickly to a feasible placement, typically at the cost of placement quality. Given the large number of fine-grained elements typically found in an FPGA device, repeated application of only localized optimization generally results in an overly inefficient final placement. This problem is especially acute for island-style FPGAs given the sparsity of routing switches and tracks prevalent in most devices. As a result, placement approaches that have a more global perspective of the design are needed to achieve desired routability.

4.1.1 Iterative Simulated Annealing

In contrast to constructive techniques, iterative placement algorithms are designed to more fully search the placement implementation space to locate the best possible placement as determined by a predefined cost function. The most common iterative technique for island-style FPGAs (and for many other design problems) is simulated annealing [50]. The simulated annealing algorithm starts with a feasible placement, created either through random assignment of design logic blocks to physical logic blocks, or through the use of constructive approaches and then repeatedly generates placement perturbations in the form of logic blocks swaps. While it clearly makes sense to greedily accept perturbations that reduce overall cost, the search aspect that makes simulated annealing unique is its treatment of swaps that increase or have no effect on overall cost.

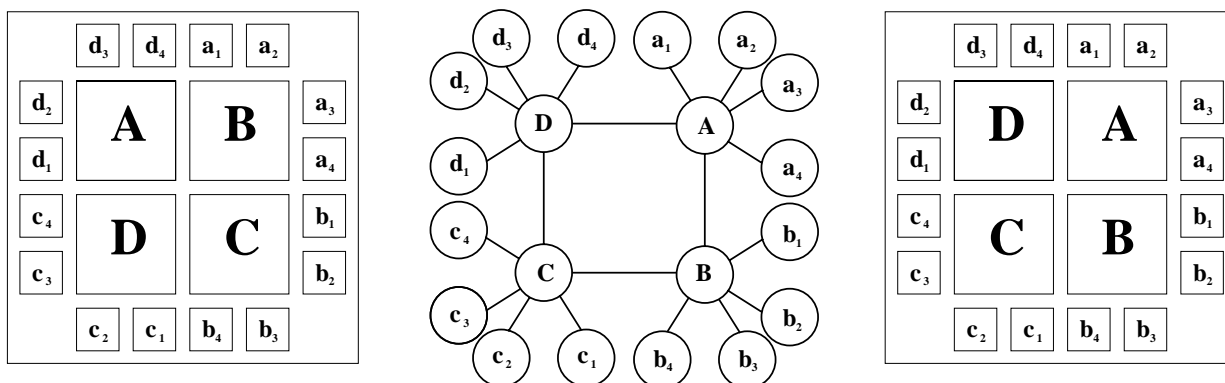


Figure 4-1: Example of a Local Placement Minima

As mentioned in Chapter 2, limiting swap acceptance for a design to only those swaps that improve placement cost tends to lead to a final placement that is far from optimal. Consider, for example², the circuit and sample placements shown in Figure 4-1. In this example, the circuit in the center of the figure is placed at left in a local cost minimum. No individual swaps of perimeter I/O pads or internal logic blocks in this configuration will result in an improvement in overall cost in terms of wire length,

¹ Feasible here indicates each physical logic block contains at most one design block.

² This example is taken from [30].

thus ending the iterative placement process if only swaps that reduce cost are accepted. This placement can be seen to be far from optimal, however, as the placement at the right clearly has lower overall wire length and is more likely to require fewer routing resources to achieve a successful route than the placement on the left. To avoid local cost minima, like the one shown in the figure, there is a need for simulated annealing to occasionally accept logic block swaps that increase overall cost. By accepting these moves, the global placement can be moved away from a local minimum enhancing the prospect that further cost-reducing swaps may find a more optimal final placement.

An important aspect of the simulated annealing algorithm is the determination of how frequently cost-increasing swaps are accepted. For most algorithms, this acceptance rate is determined based on a probability, $e^{-\frac{\Delta C}{T}}$, where ΔC is the swap cost increase and T is the *temperature*, a probability parameter which directly controls the acceptance rate. Initially, T is set to a high value so that almost all swaps, good and bad, are accepted. During progression of the algorithm, T is repeatedly reduced and fewer higher cost permutations are accepted, thus allowing convergence to a final result. Important factors that effect the run time and quality of simulated annealing algorithms are the determination of starting temperature T , adjustment of T , number of permutations attempted at each T , and the ending criteria for the algorithm. These parameters have been the subject of a great deal of research and are reviewed in subsequent sections.

Following an elaboration of each of these parameters, an experimental analysis of the relationship between the run time of simulated annealing and FPGA routability is made. This analysis is performed by running simulated annealing for various lengths of time and then evaluating placement quality in terms of overall wire length and routability. It is shown that as design sizes scale, the amount of annealing time needed to achieve the same relative quality of placement increases non-linearly.

4.2 Simulated Annealing Formulation

While individual implementations of simulated annealing subtasks, such as starting temperature determination and temperature adjustment, vary, the flow of a typical annealing formulation, shown in Figure 4-2, is constant across most implementations. The italicized subtasks in the figure have been the subject of extensive research with regard to placement for FPGAs and other design problems. For the evaluation performed in this chapter, previously-tested subtask approaches are employed to create an understanding of the growth rate of annealing time versus design size for the five benchmark circuits previously used to evaluate the growth rate of routing time relative to design size. While a comprehensive review of all previous annealing implementations is beyond the scope of this chapter, a brief review of the important issues regarding annealing is provided subsequently.

```

T = StartingT()
Moves_per_iter = MovesPerIter()
While (StoppingCriterion(T) == FALSE)
    Move_count = 0
    While (Move_count < Moves_per_iter)
        Swap blocks
        Evaluate  $\Delta cost$ 
        If  $\Delta cost < 0$ 
            Accept swap
        Else if (random(0, 1) <  $e^{-\frac{\Delta cost}{T}}$ )
            Accept swap
        Else
            Reject swap
        Move_count++
    EndWhile
    T = AdjustT(T)
EndWhile

```

Figure 4-2: Simulated Annealing Algorithm

Cost Function

An important issue for simulated annealing is the cost function used to evaluate the quality of the global placement. In most cases, including the following experimentation, the overall placement wire length, determined from net bounding boxes, is used to judge placement quality. Several other cost metrics have recently been used with simulated annealing for FPGAs with varying success. In [12], a cost function was formulated which explicitly took into account the demand versus supply of routing resources for small regions of the target device. Interestingly, it was found that this explicit evaluation of routing congestion achieved very little improvement in reducing W_{min} , the minimum channel width needed to route the design, at the cost of more than an order of magnitude additional computation time. In [41], annealed placement and maze routing were combined into a single placement formulation. In this case, the cost function for annealed placement swaps was derived from the number of unrouted design nets and the desired minimum clock period of the circuit. While this approach created layouts with about 20% better performance than the discrete layout flow of placement followed by routing, run times for even small designs of 200 logic blocks measured over 3 hours, effectively making the approach non-scalable. In [60], Trimberger mentions that in addition to wire length, Xilinx uses an alignment cost metric to stack blocks driven by high fanout signals into rows and columns so that long lines may be more efficiently used. This optimization is not used in the experimentation described in this chapter but could be added in future experiments at the cost of additional placement evaluation time.

Start Temperature

For the following experiments, the start temperature (StartingT) is determined using an approach first developed by Huang [31]. An initial random placement of all design logic blocks and I/O pads (of total count N_{blocks}) serves as a start point for evaluation. Determination of starting temperature commences by performing N_{blocks} random pairwise swaps of these blocks and pads. The initial annealing temperature is then set to 20 times the standard deviation of the cost for these moves. This design specific procedure creates an initial temperature that accepts of high percentage of swaps ($> 98\%$) in the initial stages of the annealing algorithm.

Annealing Schedule

In [58], it was determined that it is desirable to keep the percentage of swaps accepted at each temperature, R_{accept} , as close to 0.44 as possible. In [14], a flexible annealing schedule (AdjustT) based on R_{accept} was developed. Changes in \mathbf{T} based on this value are shown in Table 4.1. For acceptance rates around 0.44, the temperature is reduced by only a small amount to encourage additional searches in the current range. If most pairwise block swaps are accepted, the temperature \mathbf{T} is reduced by half. Finally, if few swaps are accepted, the temperature is reduced moderately towards a final termination point.

Fraction of moves accepted R_{accept}	α
$R_{accept} > 0.96$	0.5
$0.8 < R_{accept} \leq 0.96$	0.9
$0.15 < R_{accept} \leq 0.8$	0.95
$R_{accept} \leq 0.15$	0.8

Table 4.1: Temperature update schedule [14]

Moves Per Iteration

The ideal default number of moves at each temperature was determined in [58] to be $10N_{blocks}^{4/3}$. In [14], Betz suggests that the leading constant, 10, can be scaled to tradeoff placer run time with quality. In the following section this approach for trading off run time for quality is compared with other approaches, such as reducing the annealing start temperature and changing the annealing schedule.

Annealing Stopping Criterion

As suggested in [14], annealing is terminated when the annealing temperature \mathbf{T} is less than $0.005 \times \frac{Cost}{N_{nets}}$. Additional annealing iterations for temperatures less than this value have been shown to add little to cost improvement.

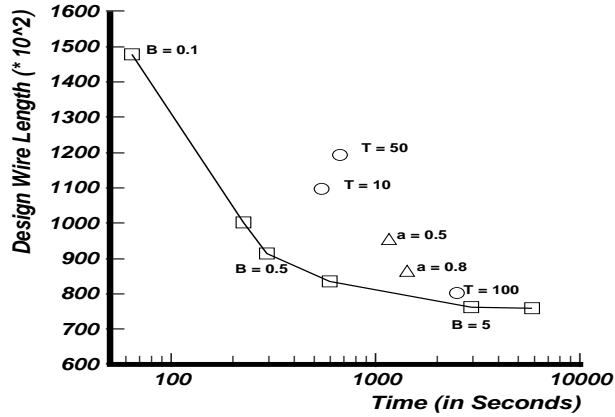


Figure 4-3: Variation of Placement Cost with Placer Run Time - Example ssp64

4.3 Experimentation

Simulated annealing has been widely studied to identify combinations of annealing parameters that create the best final quality result. Little work has been done, however, in quantifying how long it takes simulated annealing to converge to a given placement quality as design sizes scale. In this section, this growth rate is quantified experimentally for the same set of benchmarks that were evaluated in Chapter 3 with regard to wire length growth by modifying the annealed placer of the VPR toolset [14] to accept modified annealing parameter values.

Generally, longer simulated annealing runs result in improved placement quality relative to the specified cost function, in this case, design wire length. Annealing run time can be controlled by varying the annealing parameters that determine the number of logic block swaps and the swap acceptance rate. The key to evaluating the tradeoff between run time and placement quality is identifying the mix of parameter variations that result in the best run time/cost reduction curve. In an experimental evaluation, three annealing parameters, starting temperature \mathbf{T} , annealing schedule, and moves per iteration were varied over a number of designs for a set of parameter values. A representative example of the results of this evaluation for design ssp64, detailed in Table 3.6, is shown in Figure 4-3.

All run time results were obtained using a 140 MHz UltraSparc 1 with 288Mb of memory. Unless otherwise stated, annealing parameters are set to the default values discussed in the previous section. Deviations from these defaults are represented by the following graph points:

1. Trials which involve the modification of the leading constant, β , of $\beta N_{blocks}^{4/3}$ moves per iteration are represented as squares.
2. Trials with constant starting temperature values \mathbf{T} are represented as circles.
3. Trials with a fixed annealing schedule constant, α , are represented with a triangle.

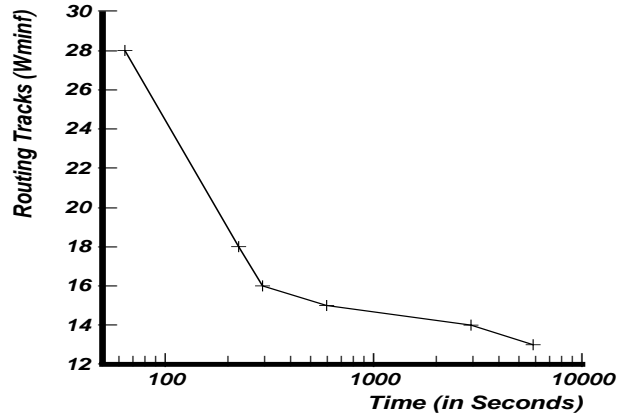


Figure 4-4: Variation of W_{minf} with Placer Run Time - Example ssp64

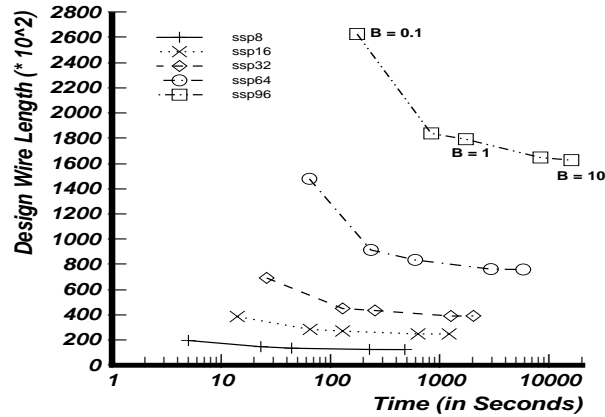


Figure 4-5: Variation of Wire Length with Placement Run Time - ssp Designs

It can be seen from the graph, that variation of the number of moves per iteration through modification of β is the most effective way to trade off placement run time for placement quality in terms of overall wire length. This observation makes intuitive sense since the starting temperature and annealing schedule are design dependent and are tuned dynamically during execution of annealing while moves per iteration remains the same across all designs. By modifying β , the search space at each temperature is narrowed, but roughly the same set of temperatures are visited as the more exhaustive case.

While evaluation of placement cost in terms of wire length gives some design quality insight, the real goal of extended placement time is reducing the amount of time needed to route a design or making design routing feasible for a target device. To measure routability in terms of track count, a second quality metric, the minimum track count that can be successfully routed in 60 seconds or less of routing, W_{minf} , is evaluated for various annealing trials. In Figure 4-4, it can be seen that the curve of W_{minf} generally follows the shape of the curve in Figure 4-3.

Now that a procedure has been established to allow for tradeoffs between annealing run time and

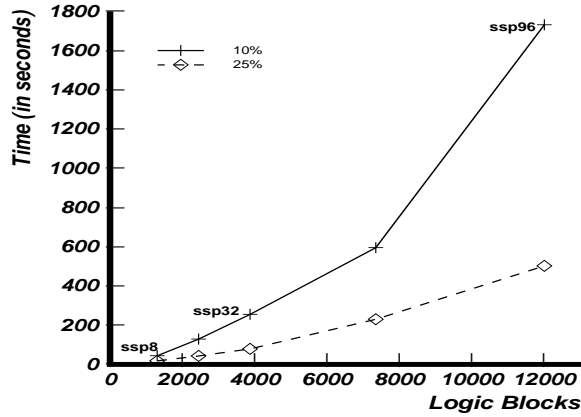


Figure 4-6: Anneal Run Time / Quality Tradeoffs

placement quality, it is possible to analyze the amount of placement time necessary to achieve a specific placement quality level across a range of designs. To perform this analysis, the five ssp benchmarks listed in Table 3.6 were placed using five different values for β , (0.1, 0.5, 1, 5, 10). The run time versus wire length results for the five benchmarks are shown in Figure 4-5. The nature of the curves indicate that increasingly long lengths of placement time are needed to achieve overall wire length results close to the best possible ($\beta = 10$) for the designs. In Figure 4-6, the result of experiments to quantify this placement time growth is shown. By varying β , each design was placed for the length of time necessary to achieve a wire length cost within 10% and 25% of the wire length cost that could be achieved with $\beta = 10$. The plots in the figure show that an increasingly large amount of time is needed to achieve these quality levels as designs sizes grow from small to large. This is not unexpected since the number of moves per iteration, $\beta N_{blocks}^{4/3}$, also increases exponentially for a given β value as the designs scale.