

List of Figures

- Figure 1.1 Standard cell and macro cell methodologies.2
- Figure 1.2 Phases of electronic system design. Adapted from [170].4
- Figure 1.3 Mapping from logical or circuit level to physical level. In c) the labels D, CLK, Q, and QB denote ports.6
- Figure 1.4 Physical design stages.8
- Figure 1.5 O -notation gives an upper bound for a function within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0 , the value of $f(n)$ always lies on or below $cg(n)$ [42].11
- Figure 1.6 Examples of directed and undirected graphs. (a) A directed graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$. Edge (2,2) is a self-loop. (b) An undirected graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (1, 5), (2, 5), (3, 6)\}$. Vertex 4 is isolated. From [43].13
- Figure 1.7 Spectrum of design methodologies for integrated circuits.15
- Figure 1.8 Example of a standard cell design. Routing has not been performed. A single bond wire is shown.17
- Figure 1.9 Example of a gate array design. The routing is not shown.18
- Figure 1.10 Example of a sea-of-gates design. The routing is omitted.19
- Figure 1.11 Example of a macro cell design.20
- Figure 1.12 Example of an island-style gate-array.21
- Figure 1.13 Examples of a) sliceable and b) nonsliceable floorplans.24
- Figure 1.14 Example of graph-based global routing for a 5 pin net. The thick line shows the Steiner tree for this net.26
- Figure 1.15 Example of plane-based global routing.27
- Figure 1.16 An example of one layer maze routing. The width or height of each square equals the grid size. The algorithm labels the grids in a breadth-first manner. Each grid was visited at the step given by the label. There are several paths from the source S to the target T avoiding obstacle O . Each path has the same length (9) but the number of bends may vary. Two such paths are shown.29
- Figure 1.17 Example of a line-probe router [89]. Escape points are labeled E.30
- Figure 1.18 Example channel and corresponding vertical constraint graph.31
- Figure 1.19 Addition of a dogleg to the example channel. The new vertical constraint graph is shown.32
- Figure 1.20 A channel routing example using two layers. This is not a left-edge algorithm[234].32
- Figure 1.21 An area route for a macro cell example using four layers.33
- Figure 1.22 Order differences in 1D compaction. Steps a-d show x -compaction before y -compaction while steps e-h show y -compaction before x -compaction. The final topology is different in the two cases. The hatched areas are on the critical path. The labels denote the constraints. A centering move strategy has been employed.36
- Figure 2.1 The basic simulated annealing algorithm.41
-

- Figure 2.2 The acceptance function for the simulated annealing algorithm.42
- Figure 2.3 Anticipated plot of the acceptance rate versus generated new configurations.44
- Figure 2.4 Typical measured acceptance rate versus generated new configurations as obtained from experiments conducted on several industrial circuits, showing the percentage of the run spent in each region of operation.45
- Figure 2.5 Target acceptance rate versus iteration.47
- Figure 2.6 Our simulated annealing algorithm. The cumulative measured acceptance rate during iteration I is A_I ; the target acceptance rate at iteration I is T_I . The algorithm will execute a total of N iterations.48
- Figure 3.1 Recursive partitioning ignores connections in other levels. At the first level, the design is partitioned into L and R. During the partitioning of L into L_1 and L_2 , pins 1 and 2 are assigned to partition L_2 . It is possible for pins 3, 4, and 5 to end up in partition R_1 .55
- Figure 3.2 Recursive partitioning using terminal propagation. The addition of pin p biases the placement of pins 3, 4, and 5.55
- Figure 3.3 The minimum rectangle that encompasses this three-pin net.63
- Figure 3.4 The bins for calculating the overlap penalty. The overlap penalty for bin b is P_b .65
- Figure 3.5 The jump in total wire length after spacing (or compaction).66
- Figure 3.6 Normalized total wire length versus final overlap penalty.67
- Figure 3.7 Overlap penalty as function of time for the ami33 benchmark circuit.69
- Figure 3.8 Results at the conclusion of stage 1 of the placement algorithm. The darkest regions are cells and the lighter shaded regions are estimated wiring areas.72
- Figure 3.9 Placement after removal of overlap.72
- Figure 3.10 The critical regions for the example circuit are shown as hatched rectangles for clarity.73
- Figure 3.11 The channel graph for the previous example.74
- Figure 3.12 An example of global routing on a channel graph. The thick line denotes the global routing tree for a single net.75
- Figure 3.13 The addition of routing tiles to the cells after global routing.76
- Figure 3.14 The example in Figure 3.13 after compaction. New critical regions are shown.76
- Figure 3.15 Original channel graph (before compaction).77
- Figure 3.16 Compaction results without channel graph constraints.78
- Figure 3.17 The topology changes. The left channel graph is the original topology. The channel graph on right is the topology after compaction.78
- Figure 3.18 Compaction results using channel graph constraints.79
- Figure 3.19 The placement topology is preserved. The left and right channel graphs are isomorphic.80
- Figure 3.20 Inaccurate modeling. TimberWolfMC version 1 overestimates the routing area needed for cell C1.81
- Figure 3.21 Placement after global routing using original wire estimator. Notice the gross inaccuracy in the estimation of the routing area for cell C1.81
- Figure 3.22 Inaccurate wiring estimation leads to poor area efficiency. White space around cells is unused area.82
-

- Figure 3.23 Placement after global routing using the statistical wire estimator.83
- Figure 3.24 Accurate wiring estimation reduces the amount of unused area. This is the minimum area placement for this example. The remaining white space does not impact chip area.83
- Figure 3.25 An example of pseudo pin placement. (a) Dark shaded areas are the macro cells. The hatched rectangles are the critical regions. The thick solid lines are channels and the dots denote a channel junction. (b) The dotted lines denote the path the global router has determined for the nets. The dark shaded areas are the macro cells.85
- Figure 3.26 The local grid lines for a placement of macro cells86
- Figure 3.27 An example of the final detailed routing.87
- Figure 3.28 Final placement and routing of *ami33* benchmark circuit.89
- Figure 4.1 An example of a mixed macro / standard cell design.92
- Figure 4.2 Flow chart of TimberWolf mixed macro/standard cell placement and routing algorithm.94
- Figure 4.3 Multiple versions of the standard cell core. User may specify additional versions.95
- Figure 4.4 Result of a partition for a mixed macro/standard cell example.96
- Figure 4.5 Result of floorplanning.98
- Figure 4.6 Tile construction and merging.99
- Figure 4.7 The final standard cell row topology.100
- Figure 4.8 The core region after standard cell routing.102
- Figure 4.9 The example circuit after placement refinement.103
- Figure 4.10 The final routing for the Texas Instruments circuit.105
- Figure 5.1 Definitions for thermal cost.109
- Figure 5.2 Definitions for isolation and proximity constraints.110
- Figure 5.3 General net classification for crosstalk versus ANAGRAM's net classification.113
- Figure 5.4 Signal waveforms for net classification. Each signal has a period of activity.114
- Figure 5.5 Definitions for crosstalk penalty.115
- Figure 5.6 Harris 5004 op-amp without any constraints. Solid bounding boxes denote output class and dashed lines denote input class. Bounding boxes grossly overlap.116
- Figure 5.7 Harris 5004 op-amp with crosstalk constraints. Solid bounding boxes denote output class and dashed lines denote input class. Crosstalk has been completely eliminated.117
- Figure 6.1 Model for interconnection of two MOS gates.119
- Figure 6.2 Example circuit.121
- Figure 6.3 Timing graph for example circuit of Figure 6.2. The nodes of the graph are the signal pins. The edges of the graph connect the pins. There are two types of edges. The thick edges are signal paths through the gates whereas the thin lines denote the signal nets.122
- Figure 6.4 Typical example of a false path. PERT analysis yields solid path. Static sensitization finds the hatched path. Dynamic sensitization discovers that both paths may propagate together. From [7].123
- Figure 6.5 Modified simulated annealing algorithm with timing analysis. After the
-

- cost has reached equilibrium, a new set of timing constraints is generated. An upper bound of M such constraints are generated for each pin pair.130
- Figure 6.6 Algorithm which builds the delay graph between sets of input-output pin pairs.131
- Figure 6.7 Algorithm for calculating the transitive fanout of the input pin. The algorithm for calculating the transitive fanin of the output pin is similar except that the queue is initialized with the output pin in step 4 and in step 7 the back edges are traversed instead. In addition, in line 8 only the outputs are visited.132
- Figure 6.8 Algorithm for removing cycles from the graph.133
- Figure 6.9 Depth-first search algorithm. Adapted from [45].134
- Figure 6.10 Algorithm for find the M shortest paths in an acyclic graph. Phase a finds the shortest path in the dag while phase b augments the number of paths using Dreyfus's method.137
- Figure 6.11 Dreyfus's M shortest path algorithm. Phase a initializes the data structures. Phase b finds the M shortest paths from the origin. Edges may be put in the priority heap once the cost is known.138
- Figure 6.12 Relaxation routines.139
- Figure 6.13 Net distribution for small circuit. Figure a is placement without timing constraints. Figure b is constrained placement.141
- Figure 6.14 Dramatic improvement in timing path distribution. The graphs plot number of paths at a given time delay.144
- Figure 6.15 Timing results for the *struct* circuit.146
- Figure 7.1 Feedthrough resources must be taken into account when building Steiner trees. A) Desired Steiner tree if Steiner point separation D is greater than average feed separation. B) Otherwise, only one Steiner point should be inserted.153
- Figure 7.2 Algorithm Overview154
- Figure 7.3 The routing tiles for a mixed macro/standard cell design.155
- Figure 7.4 Area minimization algorithm156
- Figure 7.5 Switchable segment moves. Dashed lines indicate equivalent ports. Solid lines denote the segment connecting two ports. States a, f, j, k, l are valid for row-based circuits where the row area is a keep out area (gate-arrays and standard cells). States j, k, and l will require feedthrough insertion. Additional states b, c, d, e, g, h, and i become valid when routing over the row is permissible (sea-of-gates).157
- Figure 7.6 Various Steiner trees for a 16 pin net. The Steiner tree may minimize feedthroughs, wire length, or density.158
- Figure 7.7 An example of a Steiner reconstruction move. a) the original minimum wire length Steiner tree requires two feedthroughs, one in the longest row. b) net segments which cross the longest row are removed. c) the reconstructed Steiner tree only needs one feedthrough and none in the longest row. Hence, the longest row has been shortened.159
- Figure 7.8 Cell instance may have multiple versions. Each instance has a unique pinmap. The number of ports may differ between pinmaps but the number of signals must remain constant.159
- Figure 7.9 Definitions for multiple row feedthrough assignment. A multiple row
-

feedthrough or freeway is shown at the left and a crossing is shown on the right.160

- Figure 7.10 Multi row feedthrough assignment algorithm. This algorithm assigns both macro cell feedthroughs and FPGA freeways.161
- Figure 7.11 Examples of a single row feedthrough. The feedthrough cell in the bottom row has multiple feedthrough ports.162
- Figure 7.12 Single row feedthrough assignment algorithm.163
- Figure 7.13 Removal of cell overlap. a) After explicit feeds have been added. b) After overlap has been removed.164
- Figure 7.14 An example of the maze graph construction step for a small circuit. Figure a) shows the port locations and b) displays the resulting graph.166
- Figure 7.15 Example of incremental maze routing. a) Original routes for nets and their projection onto the graph. The maximum density is 4. b) If we reroute net 1 using other regions, we reduce the density to 3. It does not help to reroute nets 2 or 3 since their ports are at critical nodes. After reroute, all ports are critical. Notice that if the new route does not contain any critical nodes, we are guaranteed that the new route reduces the density by 1.167
- Figure 7.16 Algorithm for incremental maze routing.168
- Figure 7.17 Maze route rip-up and reroute. The broken line is a segment that crosses the maximum density region. The segment is removed and the net is reconnected using the new route on the left which does not cross any maximum density regions. Hence, the total density has been reduced by one. The rerouted segment extends beyond the minimum bounding box.169
- Figure 7.18 Vertical constraint loop minimization algorithm170
- Figure 7.19 Example of freeway map.173
- Figure 7.20 Example of a Steiner tree for a FPGA. Solid lines are freeways. Dashed line is a multiple row output pin.174
- Figure 8.1 The TimberWolf graphical interface.184
- Figure 8.2 The graphics interface. *twflow* allows only one program to talk to the X server at a time. All programs share the same graphic library functions.186
- Figure 8.3 A portion of the twflow file.187
- Figure 8.4 Software implementation methodology.189
- Figure 8.5 Partial list of library modules and their function.190
- Figure 8.6 Data structure customization. The data structure has an additional field called `numpins`. The routine `init_inst` counts the number of signal pins for each instance as the data structure is built. The `<tw/structure_int.h>` set the default `USER_INST` field to the empty set. The user may redefine the fields in `<globals.h>`. In this case, the field `numpins` has been added to the end of the struct `yinstrec` definition in `<tw/structure.h>`.191
- Figure 8.7 Our coding guidelines.192
- Figure 8.8 Layout of dynamic memory. Allocation location field is only activated during debug mechanism.194
- Figure 8.9 Debug code discriminator. This function returns `TRUE` if debug label has been entered in the debug tree (set true in the `dbg` file) and debug is enabled. Notice that the coding guidelines make the function easier to read. It is evident that `debugFlagS`, `debug_treeS`, and `firstTimeS` are static variables; we will need to look at the top of this module to determine their def-
-

inition and initialization. TRUE and FALSE are macros. Yrbtree_insert is a library module - rbtree (red-black tree). The testing of the routine name is an example of defensive programming.195

Figure 8.10 Example of debug function instance. Debug label is “twsc/findcost”. This small fragment warns if the incremental cost calculation differs from an alternate method. This sanity check is enabled by the entering the line “twsc/findcost 1” in the dbg file and using the -d option on the command line to enable the debug mechanism. All debug code can be removed through the DEBUG conditional compile switch.195

Figure 8.11 Software system methodology.196

Figure 8.12 Testing methodology.198
