Contents

1 Getting started **1**
- 1.1 What is puppy? . . . 1
- 1.2 Starting puppy . . . 1
- 1.3 Interaction with puppy . . . 4
- 1.4 Exiting puppy . . . 5
- 1.5 Advanced features: Floating terminals . . . 5
- 1.6 Placement constraints . . . 5

2 Description of the algorithm **6**

3 The cost function **7**
- 3.1 Estimated wire length . . . 8
- 3.2 Routing estimation . . . 8
- 3.3 Overlap penalty . . . 9
- 3.4 Chip area . . . 9
- 3.5 Special constraints . . . 9

4 Chip boundary **10**

5 The move set **10**
- 5.1 Moves of a cell . . . 11
- 5.2 Moves of a floating terminal . . . 13
- 5.3 Moves of a net . . . 13
- 5.4 Shaking . . . 14

6 Cooling schedule **14**

7 Tuning **15**

8 Installation **16**

References

[1] M. Beards, J. Burns, A. Casotto, M. Igusa, and F. Romeo. *MOSAICO manual*, 1988.

[2] Andrea Casotto and Alberto Sangiovanni-Vincentelli. Placement of standard cells using Simulated Annealing on the Connection Machine. In *The Proceedings of the International Conference on Computer-Aided Design*. IEEE, 1987.

[3] Bruce Hajek. A tutorial survey of theory and applications of simulated annealing. In *Proc. IEEE Conf. on Decision and control*, Dec 1985.

[4] David S. Harrison, Peter Moore, Rick L. Spickelmier, and A.R. Newton. Data Management and Graphics Editing in the Berkeley Design Environment. In *The Proceedings of the International Conference on Computer-Aided Design*, pages 24–27. IEEE, November 1986.

[5] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, may 1983.

[6] Peter Moore. The General Structure of Oct. In Rick Spickelmier, editor, *Oct Tools Distribution 3.0*. Electronics Research Laboratory, UC Berkeley, 1989.

[7] Carl Sechen. *Placement and Global Routing of Integrated Circuits Using Simulated Annealing*. PhD thesis, University of California, Berkeley, 1987.

[8] Carl Sechen and Alberto Sangiovanni-Vincentelli. The TimberWolf placement and routing package. In *Proc. 1984 Custom Integrated Circuit Conference*, Rochester, may 1984.

[9] Rick Spickelmier. Oct Symbolic Policy Description. In Rick Spickelmier, editor, *Oct Tools Distribution 3.0*. Electronics Research Laboratory, UC Berkeley, 1989.

on the choice of the weights used in the cost function. The user has total control on these weights, both in the parameter file as well as interactively. This section is intended to give some advise to the novice user.

From our experience, no setting is good for all kind of circuits. Although some heuristics seem to give reasonable choices in many cases, a circuit can always be found for which those heuristics do not work.

- Given that only the relative magnitude of the weights is important, assume that the weight for the wire length is 1.
- As a rule of thumb, set the overlap weight so that at high temperatures the overlap costs about the same as the total estimated wire length.
- If you run `puppy` on the Sequent, make sure that the cluster weight is small enough, especially if the moves are selected according to their quality factors. A weight of $1E-8$ is a good start. To tune this weight turn on the debugging flag and check the quality factor for the change of ownership move at high temperatures and decrease the cluster weight until the quality factor is about tree orders of magnitude smaller than all other quality factors. This will prevent `puppy` from wasting too much time in ownership changes at high temperatures.
- If many of the cells are identical, a **grid** structure is probably the most desirable placement for those cells. It is a good idea to:
  - Use LEGAL_TRANSFORMATIONS to constraints the orientation of the identical cells;
  - Reduce the weight of nets in the cost function, and increase the area and overlap weights; by how much ?! I wish I knew, I would have coded it into `puppy`.
  - Explicitly specify the placement area (using the box in the FLOORPLAN bag).
- **Pad limited chip:** this kind of chips give problems if the core is relatively sparse. It is preferable to suggest to `puppy` the placement area (using the box in the FLOORPLAN bag). Alternatively, place the pads first and let `puppy` do the placement of the core.

8 Installation

In the directory where all the sources are type `make Ppuppy 'P=&' -P8` if you are on the Sequent, and simply `make` if you want to compile the sequential version of the program, for example on a VAX.

If everything compiles OK, you can type `make install` to install `puppy` in `~octtools/bin`.

```
for I = 1 to movesPerAnnealVar do {
    foreach annealVar in list of my annealing variables do {
        move(annealVar);
    }
}
```

Figure 4: Inner loop

at least, a number of times controlled by the parameter `movesPerAnnealVar`. We say "at least", rather that "exactly", because in the parallel version a process will continue to execute the inner loop even after it has completed `movesPerAnnealVar` iterations, until all the parallel processes have also completed that number of iterations.

The temperature used in step $k+1$ is obtained from the temperature used for step k by multiplication by a constant α whose default value is 0.9.

The annealing stops when all the following conditions are satisfied:

- the current temperature is less than `minTemperature`, which is a parameter controllable by the user;
- the cost of the placement measured at the end of the last 4 steps has not changed by more than 0.1%.

The initial temperature can be set by the user, otherwise it is automatically computed. For the automatic computation of the starting temperature, `puppy` performs a number of moves at infinite temperature (represented by -1.0), and then it computes the initial temperature based on some statistics on the cost of the performed moves.

As a rule of thumb, the initial temperature should be rather high, meaning that the acceptance ratio should be close to 1. Good values of the initial temperature vary dramatically from chip to chip.

From the menu, it is possible to set the temperature to zero, to check the behavior of the greedy algorithm, or to infinity (actually represented by the illegal temperature -1.0), to check the behavior of the totally random algorithm.

Other more sophisticated cooling schedules will be implemented in `puppy` in future versions.

7 Tuning

`puppy` is a blind annealing machine. It does not know what it is doing, it does not understand the meaning of the tradeoff between the various components of the cost function. The quality of the resulting placement is entirely dependent

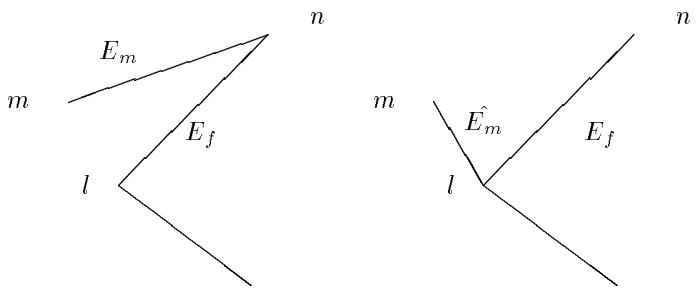

Figure 3: Example of subnet move

- Again at random, select 2 edges $E_m = (n, m)$ and $E_f = (n, l)$ incident on n; let E_m be the "moving edge" and let E_f be the fixed edge;
- The idea is to let E_m slide along E_f from n to l, so that $E_m = (n, m)$ is replaced by $\hat{E_m} = (m, l)$. This move maintains the connectedness of the graph.

Fig.3 shows an example of subnet move.

5.4 Shaking

The shaking corresponds to a small bias in the direction of the translations. A "shaking vector" of length "shakingStrength" in the `parPuppy` file is added to all translation vectors. The shaking vector rotates counterclockwise by 90 degrees every time the temperature is updated. The length of the shaking vector is internally bounded in the range between 0 and 10.

Shaking seems to have two positive effects: it helps remove overlap in the later stages of the annealing algorithm and it helps get a more squared chip by pushing the cells towards the corners of the available area.

Some more statistical study is necessary to definitely establish the effect of shaking. Set shaking strength to 0 if you want to turn off any shaking.

6 Cooling schedule

The current version of `puppy` uses a "vanilla" cooling schedule. The algorithm performs a sequence of **steps**. During each step the annealing temperature is kept constant. At each step, the algorithm can be represented as a nested loop (see Fig.4). In the inner loop, each process runs through the list of all its annealing variables and attempts a move on each of them. The inner loop is repeated,

5.2 Moves of a floating terminal

A cell with floating terminals is called a "soft" cell. In the current version of `puppy`, soft cells are limited to be rectangular. The floating terminals of a soft cell can be placed on a number of precomputed **sites** along the edges of the cell. The number of sites can be controlled indirectly with the `maxSiteCapacity` parameter in the `parPuppy` file. The `maxSiteCapacity` represents the maximum number of floating terminals that can occupy a site. The larger this number, the fewer sites will be necessary to hold all the floating terminals. On each edge of a soft cell, `puppy` will put the minimum number of sites, equally spaced, so that the number of terminals that can be physically squeezed on the edge is smaller than the combined capacity of the sites and each site has a capacity no greater than `maxSiteCapacity`.

Breaking the symmetry: If the sites are placed symmetrically around the cell, all the orientations that can be obtained by mirroring or by 180 degree rotations would be equivalent, and `puppy` would have a hard time in selecting one particular among those orientations. To break the symmetry, the left and top edge of the soft cell are assigned one site more than the right and bottom edge. This has resulted in improved finite time behavior of `puppy`, which is able to focus and improve on a particular solution earlier than it would if the soft cells were symmetrical.

Only one type of move can be performed on a floating terminal, namely the change of its site. The new site is selected at random among all the sites of the cell, except the current site of the terminal. The new site can be rejected if it is already filled to capacity or if it lies on an edge which is not legal for the floating terminal (see "TERM_FLOATING_EDGES" property). If the new site is rejected, then one of the two sites adjacent to the current site is selected. If even the second site is rejected, the move is aborted.

In parallel, two or more processors can simultaneously and independently move a floating terminal to the same site, even if the site is almost filled to capacity. In this way, it may sometime happen that the site capacity is exceeded. This condition is a benign error which in most cases is spontaneously fixed. However the condition is detected by the double-check routines and generates a warning message, which can be ignored.

5.3 Moves of a net

When the length of a net is estimated using the "subnet method", `puppy` performs an annealing also on the shape of the spanning tree of the nets. For two terminal nets there is only one possible spanning tree, so that the following discussion applies only to nets with 3 or more terminals.

The steps performed for this move are:

- At random, select a node n in the spanning tree with 2 or more incident edges;

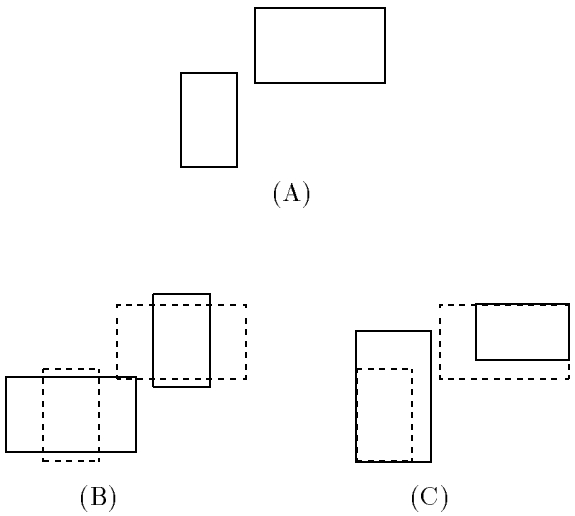

Figure 2: Types of pairwise interchange. (A): initial position; (B): interchange of the centers; (C): interchange keeping the bounding box

Ownership change: *(parallel version only)* Process p selects another process q which can accept another cell. This move does not affect the placement.

A number of flags and menu options are provided to enable or disable some types of moves.

The selection of which move to perform is at random. The probabilities of selecting a particular move are determined in one of three possible ways:

1. Constant probabilities, which are hard-wired in `puppy`;

2. Global quality factors (default); the quality factor of each type of move is the product of the expected absolute value of the cost of the move times the frequency at which that type of move is accepted. Selection probabilities are made proportional to the quality factors.

3. Cell based quality factors (not fully implemented);

The rightmost graph displays the selection probabilities for all the cell moves. The quality factors are estimated using statistics from the previous temperature step. Often these statistics are not very stable, especially for fast annealing schedules, causing the graph to appear a bit chaotic.

Each process runs through the list of its own annealing variables and performs a move attempt on each of them. In particular, during a 1-processor annealing, the annealing variables are moved always in the same order.

In the following sections the moves are described in detail.

5.1 Moves of a cell

In the following we assume that process p has decided to move a cell named A, which is one of the cells owned by p. The process can attempt any of the moves described below.

Translation: The size of the translation vector is controlled by a range limiting mechanism ([8],[7]). The *range limiter* in `puppy` is a pair of positive numbers (RX, RY). The *range limiter window* for a cell is a rectangular area of width RX and height RY centered in the center of cell. Let W and H represent the width and height of the chip boundary. Initially the range limiter is set to $(W/8, H/8)$. It is then decreased by a factor 0.7 whenever the fraction of accepted over attempted translations is less than 0.25%. The range limiter cannot become smaller than $(4, 4)$.

In a translation, the center of cell A is translated to a random position in an area determined by the intersection of the chip boundary and the range limiter window. If the intersection is empty, the cell center is placed at random inside the chip boundary. The intersection can be empty if the cell is initially outside the chip boundary or if the chip boundary itself is changed interactively, leaving the cell outside. The shaking vector (see Sec.5.4) is always added to the translation vector.

Alignment: (new move, description still missing).

Transformation: All the eight Manhattan transformations are possible for a cell, unless some are forbidden by using the "LEGAL_TRANSFORMATIONS" property. The coordinates of the cell center remain fixed during a transformation.

Translation and transformation: This move is simply a combination of a translation and a transformation.

Pairwise interchange: The cell to be interchanged with cell A is selected at random from the totality of the movable cells (pads are excluded). If the cell is owned by another process, e.g. process q, p tries to borrow it from q. If the borrowing is not possible the interchange is aborted. There are two flavors of interchanges (Fig.2), one of which involves the interchange of the centers of the cells, with or without an additional transformation. The other possibility is that of maintaining constant the bounding box of the two cells to be interchanged. The orientation of the cells is also adjusted in an attempt to exchange the aspect ratios as well.

The behavior has been chosen because it has a faster implementation and because the author believes that it is very likely that critical nets have very few pins.

Every violation has a cost specified by `conViolWeight` in the `parPuppy` file.

4 Chip boundary

The chip boundary is the area where the cells can be placed. It should not be confused with the "chip area" which is the area of the bounding box containing all the cells. In the placement window the chip boundary is represented by a dashed box and the chip area by a solid box.

`puppy` can use any one of four chip boundaries:

1. AUTOMATIC: this boundary is computed by `puppy` taking into account the total area of the cells to be placed and the desired aspect ratio; both the area of the automatic boundary and the aspect ration can be set in the parameter file or they can be changed interactively.

2. INITIAL_POSITION: this boundary is the bounding box of the cells in their initial placement.

3. FIXED_CELLS: this boundary is the bounding box of the cells which are FIXED; normally, this option is useful when there is a fixed pad frame.

4. SPECIFIED_BY_OCT: this boundary is obtained form the OCT data base; it corresponds to a box attached to the "FLOORPLAN" bag in the interface facet of the chip to be placed.

The chip boundary can be changed interactively using one of the menu entries.

Warning to micro-VAX users. The GPX's have an hardware bug that makes the X server blow up. This bug can be excited by `puppy` *if the chip boundary is very small compared to the cells. Some software protections has been added, to keep the chip boundary "large enough", but there is no guarantee that it will work in all cases. If the bug is excited you will have to remote login and kill the X-server. In extreme cases you will have to reboot the micro-VAX.*

5 The move set

Different sets of moves are possible on each type of annealing variable. A "cell" can be translated, rotated or interchanged with another cell. When `puppy` is running on the Sequent, the processes can interchange the ownership of the cells. A "floating terminal" can be moved from one site to another, a "net" with more than 2 terminals can have its spanning tree modified.

lot of work for puppy. Try to do it carefully. puppy can only compute overlap of rectangular cells. Generic rectilinear cells are expanded into rectangular cells.

The routing estimation performed by puppy is not always satisfactory. In particular, it fails in the case of abutting cells, because the expansion prevents the cells from abutting. Good results have been obtained setting "cellExpansion" to small values (1 or 2) or even to zero.

3.3 Overlap penalty

The final placement should have no overlap, but experimentation has shown that the finite time behavior of the Simulated Annealing algorithm for placement improves if overlapping is allowed in the intermediate steps of the algorithm.

The cost of overlap is proportional to the overlapping area.

In order to keep the cells inside some prescribed boundaries, an overlap penalty is computed, which is proportional to the amount of area outside the boundary. This type of overlap is more costly than ordinary overlap, by a constant factor called `chipBoundaryOverlapFactor` in the `parPuppy` file.

A small amount of overlap in the final placement does not necessarily mean that the placement is illegal, because such overlap is often due to overlap of routing regions around the cells.

3.4 Chip area

The total area of the chip is calculated as the area of its bounding box, i.e. the smallest rectangle including all the cells. If the chip contains fixed pads, they are not considered for the purpose of computing the chip area.

3.5 Special constraints

This component of the cost function is new and experimental. It can change in future versions of the program.

Sometimes one wants a critical net to be no longer than some fixed length. This is expressed in OCT by attaching the property "NET_MAX_LENGTH" to the critical net. This is an integer valued property which represents the maximum length of the net in lambdas.

puppy interprets this constraint differently depending on which method is used to estimate the length of a net. If the bounding box method is used, then puppy counts a violation every time the net has an estimated length which exceeds "NET_MAX_LENGTH". If the subnet method is being used, puppy counts a violation for every subnet which exceeds the specified length. In this way we might have the case of a critical net with many subnets, each shorter than NET_MAX_LENGTH, but such that the total length exceeds NET_MAX_LENGTH. Such situation is not penalized by the subnet method.

- f_a is the *total area of the chip* ;
- f_{ov} is the *total overlapping area* between cells; this component has to be zero at the end of the algorithm, to achieve a feasible placement;
- f_v is the total number of violations of constraints on the maximum length of some nets.
- $\alpha_w, \alpha_a, \alpha_{ov}$ and α_v are non-negative weights, that can be set for each experiment; usually, a number of experiments help determine a satisfactory set of weights. The weights are chosen by the user, using empirical criteria. See section 7 on tuning.

3.1 Estimated wire length

`puppy` estimates the length of a net in either of two ways. The first method is the classic bounding box metric, by which the length of a net is estimated as the half-perimeter of the minimum rectangle including all the terminals belonging to the net. In the second method, each net is represented using a spanning tree of its terminals. Each edge in the tree is called a "subnet". The length of a subnet is the Manhattan distance between the two terminals on the subnet, and the length of the net is estimated as the sum of the lengths of its subnets. The spanning tree of a net is not necessarily the minimum spanning tree. Instead, `puppy` performs an annealing on the shape of the tree in order to minimize its length.

See section 5 for a description of the move which allows the annealing of the spanning trees. The subnets are represented by blue segments in the placement window. Red segments are used for those nets which have only two terminals.

In terms of speed, the subnet method is intrinsically faster than the bounding box method. However, due to the need to perform extra moves to anneal the spanning trees, there is no clear speed advantage in using the new method.

The subnet method is recommended, because it is more accurate. The bounding box metric is provided only for comparison with other placement tools which also use such a metric.

One can switch between one method to the other using the "enable subnets" switch, either in the parameter file or in the interactive variable window.

3.2 Routing estimation

In `puppy` the perimeter of each cell is expanded to take in to account routing area. The expansion is depends on the pin density on each edge of the cell and on a parameter called "cellExpansion", which can be changed interactively. You can see the effect of the expansion by toggling the `show-cell-expansion` toggle in the `DISPLAY` pane of the menu. Changing the cell expansion interactively is a

nets , for which the shape of the minimum spanning tree must be determined.

In the parallel version, the set of all the annealing variables is partitioned among all the processes. In particular, each annealing variable is assigned an "owner", that is a process which has the responsibility of performing the annealing on that variable. Each process has permission to move only variables that it owns (or that it has temporarily borrowed). The current value of a variable is accessible, for reading, by all the processes. On the Sequent this is achieved by allocating the annealing variables in the shared segment of the memory.

Simulated Annealing attempts to perform a global minimization of a function defined over a finite space. In our case the finite space is the set S of all the placements and the function is a suitable "cost" which is defined by the user so that the final result has some desired properties, such as a minimum total net length, or minimum area, or a combination of the two. This characterization of the cost function is obviously faulty: it characterizes the cost function in terms of the final result which is itself influenced by the choice of the cost function, in the sense that it is one of the points of global minimum. However faulty, this characterization seems to capture what of empirical still exists in the art of using Simulated Annealing for the minimization of a complicated cost function.

3 The cost function

The cost function that `puppy` attempts to minimize is the weighted sum of four components:

- Estimated wire length;
- Overlap penalty;
- Chip area;
- Violation of constraints on length of some nets.

More precisely, let S be the set of all possible placements of the cells belonging to C_T, and define a cost function $v : S \rightarrow \mathcal{R}^+$.

The *cost function* v is the weighted sum of four components

$$v(s) = \alpha_w f_w(s) + \alpha_a f_a(s) + \alpha_{ov} f_{ov}(s) + \alpha_v f_v(s)$$

where:

- $s \in S$;
- f_w is the *estimated total wire length* ; the length of a net is estimated by computing the semi-perimeter of the bounding box including all the pins connected to the net;

- Any OCT transformation, without the prefix "OCT_": "NO_TRANSFORM", "MIRROR_X", "MIRROR_Y", "ROT180", "ROT90", "ROT270", "MX_ROT90", MY_ROT90";
- "THIS_ASPECT_RATIO" is a short hand for the list of 4 transformations which maintain the aspect ratio of the cell: "NO_TRANSFORM MIRROR_X MIRROR_Y ROT180";
- "NOT" means that all the following transformations are not legal.

Examples: "NOT THIS_ASPECT_RATIO", "ROT90 ROT270 NO_TRANSFORM".

TERM_FLOATING_EDGES is a string valued property attached to each floating terminal, specifying the edges in which the terminal is allowed to float. The default value is expressed by the empty string "" meaning that all the edges are legal. Example: "RIGHT LEFT".

FORBIDDEN_EDGES is a property attached to the "FLOORPLAN" bag in the interface of the master of a soft cell. It is an economic way to put edge constraints on all the floating pins of the cell. Example: "LEFT TOP".

2 Description of the algorithm

The input for a placement problem consists of a net list, that is a description of a set of cells and of their interconnections. The cells can be of any rectilinear shape, but rectangular cells are the most common. Each cell has a number of terminals, which are the places where the nets implementing the interconnections must be attached. The terminals can have a fixed location or they can be placed at will along some of the edges of the cell. In this second case we say that the terminals are "floating".

To describe a placement we must specify the position of each cell as well as the position of all the floating terminals. In general, a placement can be considered as a point s in a space S of all possible placements. The point $s \in S$ is specified by the values assigned to some "variables", each representing the position of a cell or of a floating terminal. In consideration of the fact that we are using Simulated Annealing to determine the optimal value of those variables, we refer to them as the "annealing variables".

The annealing variables in `puppy` are:

cells , for which a position must be selected, represented by a triplet for X and Y coordinates of the center of the cell and orientation of the cell; in future versions of the program, the shape of the cell can be chosen as well;

floating terminals , which must be placed along the perimeter of the cell they belong to, subject to some constraints;

of the cost function. See also Section 6 for more information on the cooling schedule and on the stopping criterion.

Change the placement area: The dashed box that you can see in the placement window represents the placement area. A high penalty is applied if a cell is placed outside the box. There are 4 different ways to specify the placement area, described in section 4. You can change the placement area with the menu entry *changeChipBoundary*, which rotates among the 4 possible definitions. In the case of the `AUTOMATIC` boundary, you can interactively specify total area and aspect ratio using the IV-window. When you are satisfied with the placement area, using the menu, select *recenter* to recenter the placement window.

Change expansion of the cells: (See Section 3.2). Cells are expanded by `puppy` to accommodate some routing area. You can see the effect of the expansion by toggling the `show-BB` toggle in the `DISPLAY` pane of the menu. The cell expansion can be changed using the IV-window.

1.4 Exiting `puppy`

`puppy` will stop when the stopping criterion described in Section 6 is satisfied. If you see a satisfactory placement in the placement window, you can *save-and-exit* with the menu. `puppy` creates a log file called `PUPPYdata000` containing statistical data about the annealing. You can normally discard such file.

1.5 Advanced features: Floating terminals

In a limited way `puppy` can help you do some floorplanning of a chip, in the sense that it can handle cells in which you are at freedom to choose the position of the terminals. See Section 5.2 for more details. See also the `MOSAICO` manual [1] for more details on floorplanning.

1.6 Placement constraints

The following properties can be used to specify special constraints on the annealing variables:

MOBILITY "FIXED" means that the cell's coordinates cannot be changed by `puppy`. The default is "MOVABLE".

LEGAL_TRANSFORMATIONS is a string valued property. The string is a list of words from the following vocabulary:

- "ALL" , the default values, mean that all the transforms are legal;

parameters. Try pressing some buttons on this window to see what happens. You can change the value of a variable also by typing the new value in the appropriate window. Notice at the bottom the `Confirm` button. You need to push on this button to confirm some actions that are potentially dangerous.

Plot windows: there can be 3 or more plot windows, depending on the version of `puppy` that you are running. The windows are managed by the `xg` package. There is a menu in each of these windows. The menu can be activated by pressing the middle button of the mouse. The left button is used for zooming. Press the button, hold it, drag it to define the zooming area, then release. The right button lets you make measurements on the graph. Push the button and hold it. A large cross will appear in the window, and the coordinates of its center will be printed at the top of the window. Release the button to continue.

X-event processing: the parent process in `puppy` checks the event queue after each move. Some delay in response to mouse and button actions could be experienced on very loaded machines, or on large chips if the action happens while `puppy` is performing measurements and double checks after each temperature step.

1.3 Interaction with `puppy`

You can just watch `puppy` do its job or you can try to interact with it. The most useful things you can do are :

Gently abort: you can abort because you see that the placement that `puppy` is finding is clearly unsatisfying, maybe because you have not specified correctly all the constraints, or you might have any reason you want to abort. Press any button on the console window, hold it until the menu comes, and select *exit* in the *Debug* pane. Then you will have to push the *Confirm* button in the IV window.

Adjust weights: (See section on tuning for more details.);

Adjust temperature: Simulated Annealing stops when the chip is "frozen" and the parameter called "temperature" is rather small. For a correct annealing to produce the best result **you should not play with the temperature**. However, in situations in which you are in a time crunch and want a quick solution, you will find that manually lowering the temperature will speed things up significantly. In particular, if you set the temperature to 0.0 (negative values are illegal !!) you will reduce Simulated Annealing to a more simple minded algorithm called *Iterative Improvements* which most surely will get stuck in some poor local minimum

can edit it before a `puppy` run. Any parameter in the file can also be set interactively.

- The second file is used to set constraints on the placement. This file is called either `<chip>.constraintsPuppy` or just `constraintsPuppy`. The use of this file is optional. All the constraints can be represented by OCT properties attached to instances and to nets. If such properties are already present in the chip to be placed, you do not have to use the file. This file provides a simple mechanism to enter these properties in the OCT database. In particular you can set `MOBILITY` and `LEGAL_TRANSFORMATIONS` for each instance, and `NET_CRITICALITY` and `MAX_NET_LENGTH` for each net. All of these properties are described later in this manual and also in the *Oct Symbolic Policy*document.

  This file can be created by using the `-l` option. Use `-l C` if you are only interested in constraining the placement of the cells, `-l N` if you want to care about the criticality of the nets, or `-l B` if you are going to constrain both instances and nets. Once the file is created you can edit it. The file is self-explanatory.

Suppose that the chip that you want to place is `chip:unplaced`. Type

```
puppy -o chip:placed chip:unplaced
```

First, `puppy` copies the input view `chip:unplaced` into the output view `chip:placed`. From now on, the input view will not be touched. Then `puppy` begins to read the `chip:placed`. If the chip is large, with many instances of many different masters and many nets, reading the view can take quite some time.

If you are running under X and the environment variable `DISPLAY` is properly set, `puppy` opens the following windows on your display:

Placement window: this window shows the current placement being considered by `puppy`. The window has an active menu, which can be called up by pressing and holding any button while inside the window. Use this menu to recenter the placement area in the window, to decide whether the nets should be displayed, to see which processor owns which cell, to evaluate the expansion of the cells.

Console window: this window shows the components of the cost function, the speed of the algorithm measured in moves per second and the number of processors being used. Other information might also be displayed, depending on the version you are actually running.

Interactive Variables window: this window allows the user to control most of the annealing parameters. This window is implemented by the `iv` package. There are active buttons in the window, to change the values of the

[b]

```
Placement Of MacroCells using Simulated Annealing
VERSION 4.4 With ACS : Tue Feb 28 14:34:40 PST 1989
SEQUENTIAL VERSION
puppy [options] cell-name:view-name
  -o cell:view    Specify output cell and view
  -O number       Overlap     Weight in SA cost function [1.00]
  -W number       Wire Length Weight in SA cost function [1.00]
  -A number       Area        Weight in SA cost function [1000.00]
  -C number       ConstrViolations Weight in SA cost function [1.00]
  -T number       Starting Temperature for SA [automatic]
  -R number       Seed for the random number generator
  -S number       Cooling Schedule (0,1,2)
  -a number       Moves per annealing variable [20]
  -x number       Set expansion factor for the cells [2.00]
  -c              Compute the cost of current placement and stop
  -f              Place only floating terminals
  -q              Disable the translations
  -t              Disable the transformations
  -g              Turn OFF graphic output
  -l [CNB]        List C_ells or N_ets or B_oth. Also writes parPuppy file
  -d              Enables debug
(Command line options override the values in the parPuppy file)
```

Figure 1: puppy usage

puppy reference manual

Andrea Casotto
Dept. of Electrical Engineering
University of California, Berkeley

March 5, 1990

1 Getting started

1.1 What is puppy?

puppy is a tool for placement and floorplanning of integrated circuits in the macro cells design style. The program can also be used for placement of components on a PCB. The algorithm used by puppy is Simulated Annealing ([5], [3], [2]).

The program originated [2] on the Sequent Balance 8000, a multiprocessor with shared memory. On the Sequent, puppy can use as many processors as desired, with almost linear speedup. The program can also run on sequential machines (VAX, SUN). Sequential and parallel versions are essentially identical from the user's point of view.

The intention of the author was to make puppy an easy program to use. A friendly user interface has been developed, with graphics and menus. It is highly recommended that puppy is run under an X-server, even if the program can run on an alphanumerical terminal.

1.2 Starting puppy

If you type the command puppy without any argument you should get something that looks like Fig.1. This tells you that puppy is alive and well.

puppy is interfaced to the data base OCT ([6],[4]) and the data for puppy must conform to the *Oct Symbolic Policy*[9]. If you are not familiar with OCT, you are out of luck.

puppy uses two auxiliary and optional files:

- the first file is used to set the parameters of the annealing. puppy looks first for a file called <chip>.parPuppy. If such file is missing, puppy searches for parPuppy. The file can be created by puppy itself, using one of the menu commands or using the -l option. The file is self-documenting. You