An extractor for hierarchical symbolic layouts

THE-RC 59994

Eindhoven University of Technology
Computing Centre Note 23

An extractor for hierarchical symbolic layouts

Authors: T. Bidlot and M. van der Woude

December 1984

TABLE OF CONTENTS

Abstract

An outline of a circuit extractor is proposed, which can be used for extracting VLSI circuits from a hierarchical symbolic layout description. Use is made of an interface file in which the interface between layout and circuit description is stored for each hierarchical building block. The hierarchy of the circuit description can be expanded partially or wholly to let the extracted circuit description have the hierarchy as required by the user. An outline of the algorithm on which the extractor is based is discussed. Results of a preliminary version are presented.

1. Introduction.

In this paper we propose a circuit extractor which is intended for the extraction of circuits from hierarchical symbolic layout descriptions. It is a well known fact that hierarchical design methods offer great savings in time for the designer and result in a reduction of processing time and storage use. However very often what is called a 'hierarchical design method' is no more than a hierarchical description method [2], because only an informal hierarchy is used which has to be expanded as soon as exact knowledge of the boundaries of the building bricks is required, e.g. with design rule check and with circuit extraction. In our layout design system we maintain a strictly formal hierarchy which enables not only the designer but also computer programs to consider hierarchical compound cell instances as independent units which interact with the outside world only via a well defined interface consisting of a domain and a set of terminals. Hierarchical design rules preserve sufficient separation between different layout components, such that parasitic effects are kept below a harmless treshold. Since the layout description reflects this hierarchy it is a simple task for a circuit extractor to translate a layout description to a circuit description. All layout components except wiring can be extracted straightforward bye translating their layout description to the corresponding circuit description. We will assume that the only parasitics that cannot be discarded and that therefore have to be extracted by geometric search, are the parasitic capacitances of the wiring to the substrate. The same geometric search can provide the interconnection pattern between the layout components.

The goals for the circuit extractor presented are to provide the designer with a useful interactive tool with all the usual properties of a circuit extractor. Because of its hierarchical method it will require very few computer resources. Compared to other hierarchical extractors, see e.g. [4] its new feature is that the interface of a compound cell between its layout and circuit description may, but need not, be prescribed by the user, in order to the user or a verification program to compare extracted circuit descriptions with other descriptions, e.g. those used previously for simulation. If the interfaces where not the same , i.e. parameter positions do not correspond, this is not so easy.

In what we call a hierarchical layout a compound cell instance may be regarded as a building brick, consisting of a domain, which defines a forbidden region, and a set of terminals at which interconnections to the interior of that cell can be made by overlap and by abutment. In the NMOS process, which we shall assume, the interconnections are restricted to the metal, poly and diffusion layers. The domain may consist of several, not necessarily overlapping, rectangles in these layers. To design

such a layout we use an editor [1] which can operate in a symbolic and in a geometric mode. In the symbolic mode compound cells may be designed within a fixed grid. These symbolic compound cells are composed of the following symbols:

- rectangular pieces of wiring
- rectangular via cells
- transistors with rectangular active area
- instances of previously defined compound cells
- instances of geometric cells

In the geometric mode, which is primarily intended for the lower levels in the design hierarchy, geometric cells may be build up of any combination of rectangles and previously defined cells.

The definitions of a cell in the layout description are marked as follows:

- s (symbolic compound cell)
- g (geometric cell)

The definitions of transistor and via symbols are marked as:

- et (enhancement transistor)
- dt (depletion transistor)
- v1 (metal/diffusion via)
- v2 (polysilicon/diffusion via)
- v3 (metal/plysilicon via)

Within the extractor symbolic compound cells or shortly compound cells are considered as non leafcells. All other symbols are considered either as leaf cells (marked g, et, dt) or as wiring (marked v1, v2, v3 or not marked). It is assumed that the output of the extractor is a description for SPICE or another circuit analysis program. Such a description may have the same hierarchical structure as the layout or it may be (partially) expanded.

Instances of leafcells in the layout description marked g are translated to subcircuit calls whose definition has to be provided by the user or by a geometric extraction program. Via cells and pieces of wiring are translated to nets which have parasitic capacitance to the substrate determined by the total effective area and the perimeter (to include sidewall capacitances). Transistor cells are translated to transistor calls with length and width as geometric parameters. Additional parameters, e.g. to describe the source/drain geometry may be extracted if desired.(A SPICE MOSFET may have up to 8 geometric parameters). The user has to provide two model descriptions one for enhancement and one for depletion transistors. The definition of a symbolic compound cell is inspected by the extractor and either translated to a subcircuit definition or expanded to a lower hierarchical level if

the user specifies so.

We will assume that the layout description has passed a design rule checker, so overlap or abutment is always sufficient to make valid contact and separation rules are always satisfied.

2. Requirements.

We require that the extractor can be used highly interactive. The user should be able to specify:

- Which output language (in case more than one output languages are supported).
- Extraction with or without parasitic capacitances.
- Compound cells to be extracted. Only the highest hierarchical levels need be specified. If the extractor discovers a call to a lower level compound cell this one has to be extracted "on the fly".
- Compound cells to be expanded. A user may wish to consider the circuit description of a design with another hierarchy than the layout description. This can happen e.g. if the layout designer has added one or more additional hierarchical levels for ease of manipulation. To compare the extracted circuit to the input circuit for the layout design, these additional hierarchical levels should be expanded, i.e. replaced by their contents. Expansion is also required when the simulation package does not support hierarchical input.

- Compound cells to be considered as leaf cell. A user may wish to specify compound cells as leaf cell, e.g. if their subcircuit definition is already present.

To guide the extractor in producing a circuit description the user should be able to specify the layout/circuit interface of a compound cell. This consists of:

- The compound cell name and the corresponding subcircuit name. In most cases it will suffice to use as a subcircuit name the name of the corresponding name of the compound cell definition. This will be the default case. However in some cases there may be reasons to deviate from this convention, e.g. when more than one layout cell correspond to the same subcircuit.

- A list of layout terminals with for each layout terminal the corresponding subcircuit contact. Note that terminals may be internally interconnected, this implies that more than one terminal may correspond to the same subcircuit contact. If the layout description does not contain names for some terminals then the unnnamed terminals can only be characterized by either their position in the data structure or by their geometric

position. The user may (but need not) wish to specify the terminal correspondence. If e.g. a subcircuit definition has to be used it is required that the terminals correspond with subcircuit contacts in a prescribed way.

For convenience of the user the layout/circuit interface should be stored as a file, the interface file, which is maintained by the extractor. By comparing timestamps the extractor can decide whether a stored interface description of a compound cell is still valid.

The output of the extractor has to be a textfile containing a series of one or more subcircuit definitions in the input language specified by the user. From the requirments mentioned it follows that the extractor can be operated from a terminal without graphics facilities and that it does not need to modify the input layout file.

3. Main algorithms.

Some details of the algorithms depend on the desired output format. For simplicity we will assume that a SPICE description has to be produced. By means of a dialogue with the user the following lists are created:

- EXTRACT_COMPOUND_LIST

This list contains compounds to be extracted in order of non-decreasing hierarchical level. A compound is represented by the pointer to the start of its definition in the layout file, together with a boolean EXPAND which is true when the compound has to be expanded.

All instances of compound cells met during extraction that do not appear in EXTRACT_COMPOUND_LIST will be considered as leaf cells.

- SUBCIRCUIT_LIST

This list contains at the end of the dialogue a record for each leaf cell. During the extraction process it is extended with descriptions of subcircuits that are extracted. The description of leaf cells is obtained from their interface as it is present in the interface file. The description of a subcircuit consists of:

NAME - subcircuit name
NFORMN - the formal nets are numbered 1 .. NFORMN
NLOCN - the local nets are numbered NFORMN + 1 .. NFORMN + NLOCN (NLOCN = 0 for leaf cells)

The following lists, that are empty for leaf cells, are also part

of a subcircuit description. They are constructed for cells to be extracted during the extraction process.

WIRING_LIST- contains for each net the area and perimeter in each of the layers

SCALL_LIST - contains for each subcircuit call a pointer PSDEF to the definition of the corresponding subcircuit in SUBCIRCUIT_LIST and a pointer PPAR to PARAM_LIST, an array which contains for each call a list of actual net parameters

TCALL_LIST - has an entry for each transistor call containing transistor type (enh. or depl.), net parameters and geometric parameters

EXPAND - boolean which is true when the subcircuit has to be expanded

If it appears that one of the leafcells does not have a valid interface in the interface file or if the user wants to change it, the interface will be interactively established or updated and stored in the interface file. During the extraction process the interface file is updated in the same way with the interfaces of the compounds in EXTRACT_COMPOUND_LIST.

The extraction process may be described by the following program (Algorithm 1).

```
PROCEDURE EXTRACT (EXTRACT_COMPOUND_LIST,
                   SUBCIRCUIT_LIST      );

BEGIN
   WHILE EXTRACT_COMPOUND_LIST NOT FINISHED DO

      GET (COMPOUND from EXTRACT_COMPOUND_LIST);
      CREATE (at pointer PSUB a new SUBCIRCUIT record);

      WITH COMPOUND, PSUB DO

         CONSTRUCT (WIRING_LIST, SCALL_LIST, TCALL_LIST);
{
      Construct employs a SCANLINE algorithm [3] to determine the
      nets and their geometric properties. It is taken into account
      that nets may be interconnected by overlap , abutment and
      with vias and also via internal interconnections of
      subcompound instances which can be detected from the
      interface description.
}

         ESTABLISH interface and store it;

{
      As soon as the wiring has been determined it is possible to
      establish the interface, since now it is known which sets of
      formal terminals are interconnected within the compound and
```

```
      hence  correspond to the same net parameter. If the interface
      is not present a default interface will  be  generated.  The
      user  has  the  possibility  to  adapt  the  interface to his
      wishes.
}

         ADJUST ( according to interface:
                  NAME, WIRING_LIST, SCALL_LIST, TCALL_LIST);

       END {WITH};

       APPEND (to SUBCIRCUIT_LIST, record PSUB);

       IF  NOT PSUB.EXPAND  THEN
            EXTRACT_SUBCIRCUIT (PSUB)
       FI;
     END {WHILE}

END {EXTRACT};
```

Algorithm 1.

The procedure EXTRACT_SUBCIRCUIT is described below (Algorithm 2), it uses a recursive procedure EXTRACT_BODY which is described as Algorithm 3.

```
PROCEDURE EXTRACT_SUBCIRCUIT (PSUB);

BEGIN
   WITH PSUB DO

       {write heading}
       WRITE('.SUBCKT ', NAME, 1 TO NFORMN );

       LACTW := NFORMN + NLOCN;

       FOR I := 1 TO LACTW DO
          ACTW [I] := I;
          CAPACITY [I] := CAP_FUN (WIRE_LIST [I])
       END {FOR};

       NWUSED := LACTW; { NWUSED is # of wires used}
       NSC := 0; { NSC is # of subcircuit calls extracted}
       NTC := 0; { NTC is # of transistor calls extracted}

       EXTRACT_BODY (ACTW, CAPACITY, PSUB, NSC, NTC, NWUSED);

       {write capacity}
       FOR I := 1 TO NWUSED DO
          WRITE_CAPACITANCE ( CAPACITY [I] );
```

```
        END {FOR};
        {write end}
        WRITE (´.ENDS ´, NAME);
     END {WITH}
  END {EXTRACT_SUBCIRCUIT};
```

Algorithm 2.

```
PROCEDURE EXTRACT_BODY (ACTW, CAP, PSUB, NSC, NTC, NWUSED);

BEGIN
   WITH PSUB DO

      WHILE TCALL_LIST NOT FINISHED DO
         { write transistor call}
         NTC :=NTC +1;
         WRITE_TCALL (next element from TCALL_LIST);
      END {WHILE};

      WHILE SCALL_LIST NOT FINISHED, WITH NEXT ELEMENT DO

         {fill ACTW1 with actual numbers of formal wires}
         FOR I := 1 TO PSDEF.NFORMN DO
             ACTW1 [I] := ACTW [PARAM_LIST [PPAR + I]];
         END {FOR};
         IF NOT EXPAND THEN  {write call}
            NSC := NSC + 1;
            WRITE (´X´, NSC, FOR I := 1 TO PSDEF.NFORMN DO
                        ACTW1 [I], NAME);
         ELSE

            {assign actual numbers to local nets }
            FOR I := 1 TO PSDEF.NLOCN DO
               NWUSED := NWUSED + 1;
               ACTW1 [PSDEF.NFORMN + I] := NWUSED + 1;
               CAPACITY [NWUSED] :=  CAP_FUN
               (PSDEF.WIRE_LIST [I+PSDEF.NFORMN])
            END {FOR};
            {add capacity to actual nets}
            FOR I := 1 TO PSDEF.NFORMN DO
               CAPACITY [ACTW1 [I]] := CAPACITY [ACTW1 [I]] +
                        CAP_FUN (PSDEF.WIRE_LIST [I]);
            END {FOR}

            {extract body}
            EXTRACT_BODY (ACTW1, CAP, PSDEF, NSC, NTC, NWUSED);
         FI
      END {WHILE}
   END {WITH}
END {EXTRACT_BODY};
```

Algorithm 3.

4. Calculation of capacitance.

The parasitic capacitance is calculated from the geometric data in WIRE_LIST by the function CAP_FUN. This function calculates the sum of the area capacitance and the sidewall capacitance of a wire in each of the layers in which the wire is present. The geometric data area and perimeter for each layer are calculated during the SCANLINE search. Since only capacitance to the bulk is calculated, account has to be taken of wires of the same net on top of each other [4]. In that case the capacitance of the top wire to bulk will be assumed zero.

Note that, since always one hierarchical level at a time is considered, it is not possible to make this correction if wires of the same net are on top of each other at different levels of the hierarchy. So if e.g. a metal linepiece runs across an instance of a compound call just on top of a diffusion line piece to which it is connected via the terminals than the capacitance of both pieces will be added to the capacitance of the net, resulting in a too large parasitic capacitance. When calculating capacitances the parts of wires that overlap with actual contacts are ignored. So if a wire connects a formal contact (terminal of the definition) to an actual contact (terminal of an instance) then the capacity of this wire is calculated as if it consists of the area of the formal contact and all area outside the actual contact. Area and perimeter of actual contacts are assumed to belong to the internal of the instance.

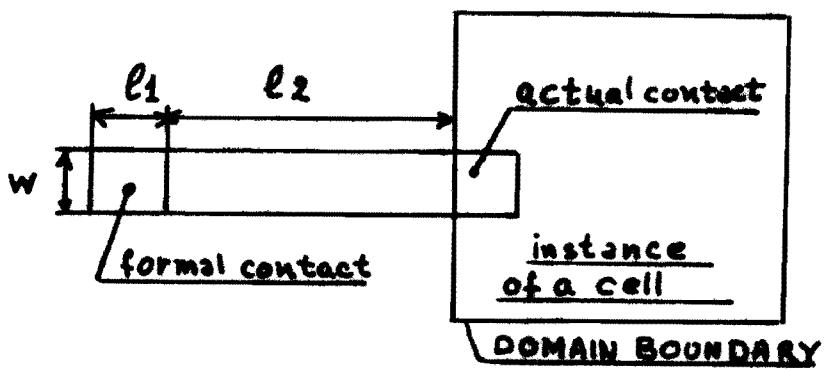

fig. 4.1 Calculation of wire area and perimeter.
Area = w * (l1 +l2)
Perimeter = (w + l1 + l2) * 2

E.g. in fig. 4.1 the capacitance is calculated as:

$$C = Ca * w * (l1 + l2) + Cb * (l1 + l2 + w) * 2$$

With Ca and Cb the capacitance per unit area, resp. per unit perimeter. With this calculation the abutting part of the perimeter will be counted twice, while it is not part of the perimeter. The result is a capacitance which is slightly too large. We could avoid this error by always omitting the abutting part of terminals.

Other capacitances.

It is also possible to extend the program with calculations of other types of parasitic capacitances. However our approach does not admit us to calculate coupling capacitances between nets at different hierarchical level, since these capacitances disturb the hierarchy. If this kind of parasitics would effect circuit behaviour seriously, the only possibility is to forbid wires across the domains of compounds and to make design rules for domain separation and for separation of domains and layout elements so conservative that coupling may be neglected.

5. Results.

The method outlined above has been implemented provisionally in a Fortran program, no recursion is used and no use is made of interface files. Further sidewall capacitances are ignored. Of this tentative program we present some results. In fig. 5.1 a circuit of a D-flipflop is shown consisting of 6 NOR-gates.

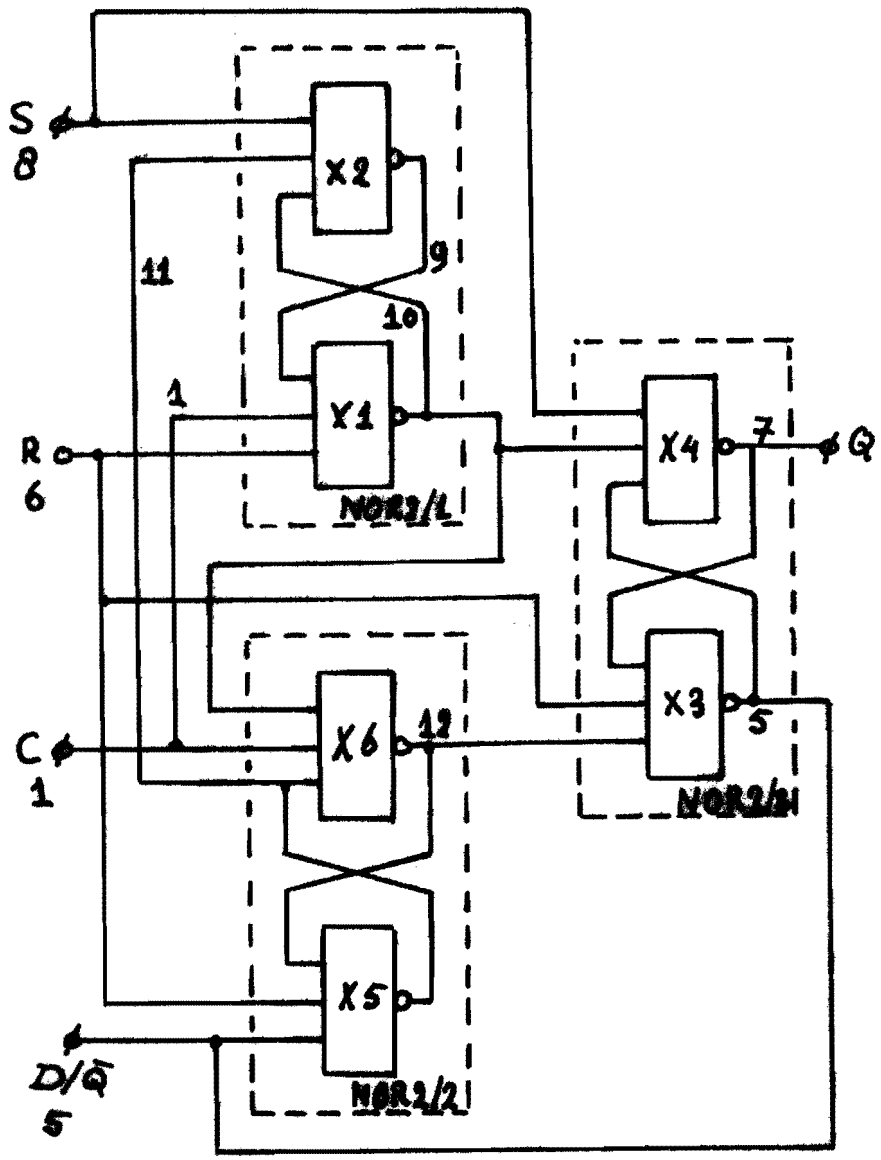

fig. 5.1 Circuit diagram of a D-flipflop with 6 NOR-gates.

The layout of this circuit has been constructed with our hierarchical layout editor. For ease of manipulation the hierarchy was extended by combining 2 NOR-gates to a building brick of type NOR2. The hierarchical layout is shown in fig. 5.2

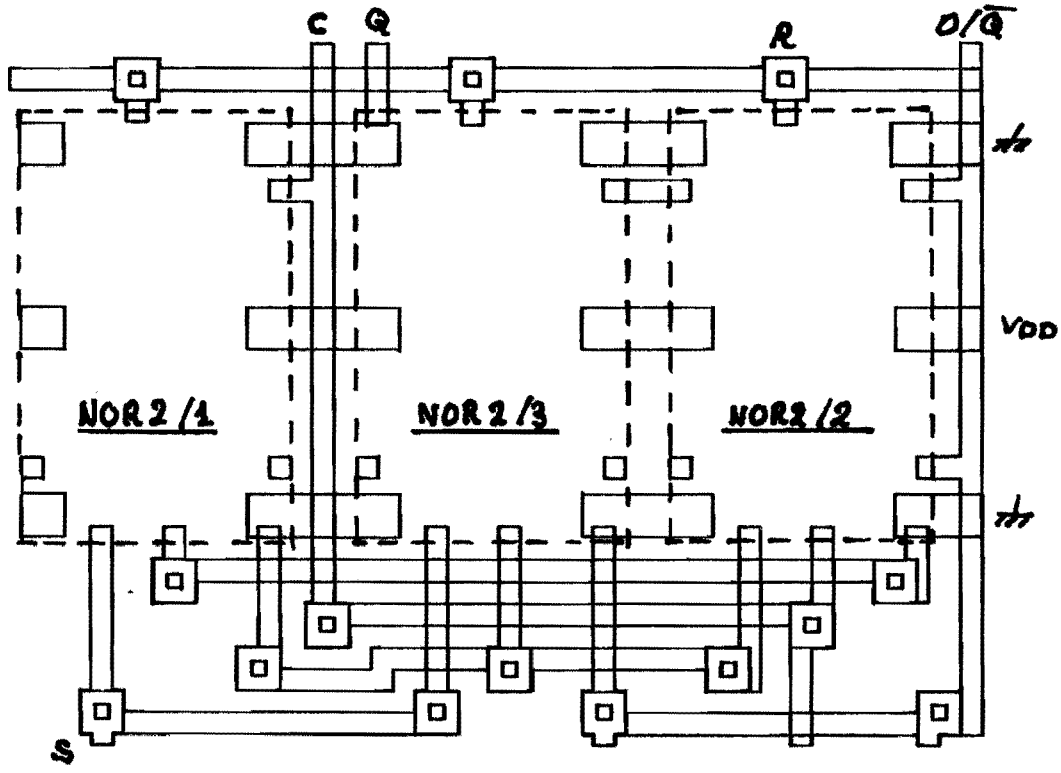

fig. 5.2 Hierarchical layout of D-flipflop using
3 building bricks of type NOR2.

The layout of a NOR2 block and of a NOR block is shown in fig. 5.3.

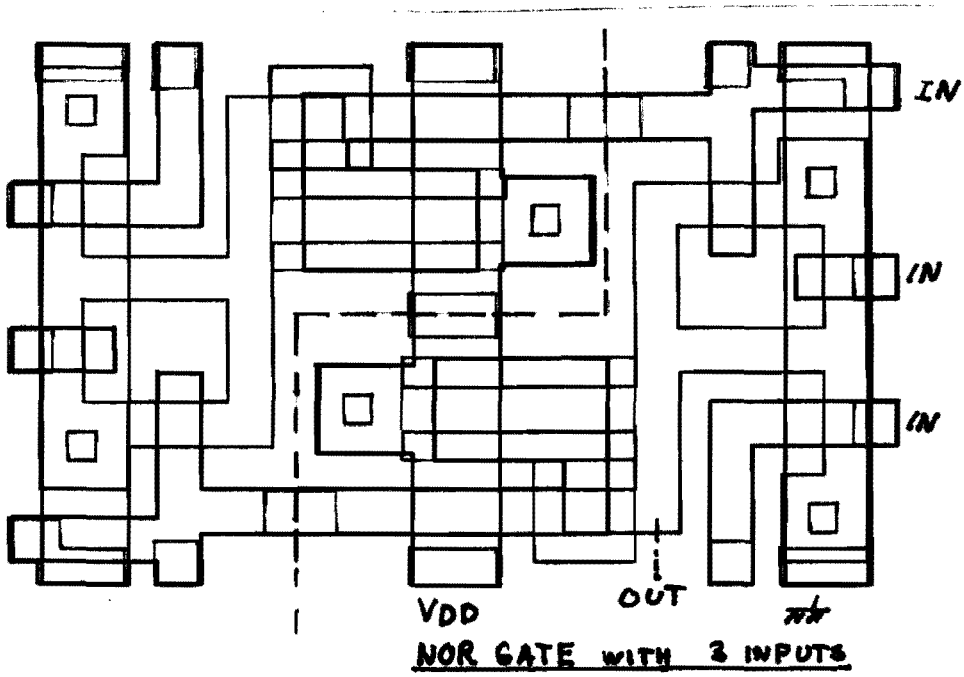

fig. 5.3 Detailed layout of a NOR2 building brick consisting of two NOR-gates.

The result of the extractor is the following SPICE description, in which the NOR2 compounds have been expanded:

```
.SUBCKT NOR  1 2 3 4 5 6
C1 1 0 23.0FF
C2 2 0 36.7FF
C3 3 0 16.9FF
C4 4 0 35.4FF
C5 5 0 10.8FF
M1 4 4 1 0 NMOSTD W= 6U, L= 24U
M2 4 3 2 0 NMOSTE W= 6U, L=  6U
M3 4 6 2 0 NMOSTE W= 6U, L=  6U
M4 2 5 4 0 NMOSTE W= 6U, L=  6U
.ENDS NOR

.SUBCKT FLIPFLOP 1 2 3 4 5 6 7 8
C1   1   0   121.0FF
C2   2   0    14.0FF
C3   3   0    14.0FF
C4   4   0    14.0FF
C5   5   0   123.8FF
C6   6   0    86.8FF
C7   7   0     6.8FF
C8   8   0    70.0FF
C10  10  0    88.0FF
C11  11  0    61.6FF

X1   3   4    9   10    1    6   NOR
```

```
X2   3   2   10   9    8   11   NOR
X3   3   4    7   5   12    6   NOR
X4   3   2    5   7    8   10   NOR
X5   3   4   12   11   5    6   NOR
X6   3   2   11   12  10    1   NOR
.ENDS FLIPFLOP
```

The extracted circuit has been simulated with and without parasitic capacitances. The version with parasitics was about a factor 2 slower.

6. Future extensions.

The extractor outlined above can easily be extended with a circuit verifier. To be able to verify extracted circuits with existing descriptions we have to provide with the interface description the classes of logically equivalent parameters and groups of parameters which are logically equivalent [5]. The extraction of parasitics can be extended to resistors and coupling capacitances and others, provided the hierarchy is not disturbed. Further it is desired to extend the extractor with the possibility of the extraction of geometric cells.

7. References.

1. A. H. D. Bidlot e.o., "Editor for Hierarchical Structured Layouts of large Integrated Circuits (VLSI-circuits)", Proceedings of the NGI-SION symposium, Amsterdam, 16-17 april 1984, pp. 151-160.

2. C. Niessen, "The role of CAD tools in VLSI design methodology", ESSCIRC Digest of Technical Papers, Freiburg, Sept. 22-24, 1981, pp. 75-86.

3. Bentley J. L., Haken D. and Hon R. W., "Statistics on VLSI Designs", Report CMV-CS-80-111, Dept. of Comp. Science Carnegie Mellon University, Pittsburgh, April 17, 1980.

4. Tarolli G. M. and Herman W. J., "Hierarchical Circuit Extraction with detailed parasitic capacitance", Proceedings 20th Design Automation Conference, 1983, pp. 337-345.

5. Losleben P. and Thomson K., "Topological Analysis for VLSI Circuits", Proceedings of the 16th Design Automation Conference, June 1979, pp. 461-473.