An interactive graphics editor for a schematics entry program

de Craen, A.J.M.

Award date:
1988

5533

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
DESIGN AUTOMATION SECTION

AN INTERACTIVE GRAPHICS EDITOR

FOR A SCHEMATICS ENTRY PROGRAM

A.J.M de Craen

Master thesis
reporting on graduation work
performed from october 1987 to august 1988
by order of prof. dr. ing. J.A.G. Jess
and supervised by ir. G.L.J.M Janssen

ABSTRACT

For a schematics entry program, an interactive graphics editor is developed.
The editor program has a very flexible structure. Parts of it can be changed without affecting the rest of the program.

The program is user-friendly. It has some attractive features like pop-up menus, a repeat button and rubberbanding.
Furthermore, a help facility is implemented.

The datastructure is graphics orientated to optimize response time.

Device dependency is restricted to a graphics library which is built upon routines provided by a graphics package.

1. INTRODUCTION

1.1 SCHEMATICS ENTRY

Due to growing complexity of Integrated Circuits CAD tools are developed as an aid to IC designers.
A schematics entry program is a CAD tool for *network database* manipulation. A network database contains the description of an electrical circuit expressed in some network description language. The network description is used as input to other CAD tools like simulators, automatic placers and routers and layout generators.

The ICD project (Integrated Circuit Design) is a cooperation of three Dutch Universities of Technology, ICS (a Dutch software company), British Telecom and PCS (a German workstation manufacturer), financially supported by the Commission of the EEC.
Its goal is the development of an open system of CAD software tools for VLSI design.

ESCHER (Eindhoven SCHematic EditoR) is a schematics entry program developed at the Eindhoven University of Technology as a result of the ICD project. ESCHER is an interactive graphics editor which provides the means for creating hierarchical structured circuits by manipulating their components. The output generated by ESCHER is in accordance with the ICD network standard, interfacing with several simulation programs and layout generation tools.

1.2 ESCHER

In ESCHER the basic unit of information is a *template*, having an inside and an outside.
The outside of the template, called its representation, consists of graphical elements (*terminals, lines, circles* and *arcs*) and non-graphical elements, called formal parameters. Both the terminals and the formal parameters contain network information, whereas the lines, circles and arcs have no meaning but pictorial.
The inside of the template, being the electrical circuit, consists of graphical elements, all having network information. These elements are *wires, system terminal ports (*or *terminals)* and *subcircuits*. A subcircuit is an invocation of another template. An invocated template is called an *instance* and is shown in the circuit by the representation of the invocated template. Terminals are the contactpoints of (sub)circuits with the surrounding world. Interconnections between subcircuits is established in two ways: either by abutment (when the terminals of the subcircuits coincide) or by defining wires between the subcircuits.
The terminals of the representation and the terminals of the circuit must match. They are assumed to be electrically equivalent.
With ESCHER libraries of predefined templates can be used for ease of creating new templates.
Furthermore, a verify command is provided to check network information.
ESCHER provides both the bottom-up and top-down design strategies. With the bottom-up design strategy instances having no insides (called *leafcells*) are created first. The leafcells are used for the creation of the templates of the next level and so on. Top-down design methods start at the highest level. At each lower level the circuits of the instanced templates are refined.

1.3 ESCHER+

The extension of ESCHER with an event-directed simulator resulted in ESCHER+. In ESCHER+ a template can have a behaviour description, expressed in a LISP-like behaviour description language.
When simulating a circuit ESCHER+ derives the circuit behaviour from its subcircuit behaviour descriptions and subcircuit interconnection information. ESCHER+ uses animation as a means to illustrate circuit behaviour, thus visualizing signal flow through the circuit.
The program is written in C and runs in an Unix environment on HP900 computer systems and APOLLO workstations.

To describe the circuit behaviour in LISP and to make use of the facilities offered by the windowing system APOLLO supplies on their workstations, ESCHER+ is completely rewritten. The work is divided into two parts: H. Fleurkens did the network and behaviour part [7], I did the graphics part.

2. EDITOR CONCEPT

A schematics entry program can be divided into two major components: a *network manager* and a *graphics editor.*
The network manager creates and updates both the network description of the circuit and the behaviour description of the subcircuits. The network description can be extracted from the objects of the schematic.
The editor is an interactive graphics program for the design of schematics, called *templates.*
A template has two appearances, called *symbol* and *contents.*

The symbol (or representation in ESCHER) of a template provides the outer view of the template. It contains elements which make up the representation of the template. Currently the only available elements are *lines* and *circles.* Future extensions could include elements such as *arcs* and *text.*
Symbols are used when instancing the template in the circuit of another template. Instancing of templates allows the creation of hierarchical structured circuits.

The contents (or circuit in ESCHER) of a template is the inside of the template, consisting of the following objects:

- *connections (or wires)*
- *contacts (terminals)*
- *instances (or subcircuits)*

A terminal is visualized by a triangle having a direction (either *north, west, east or south*). The direction of the terminal indicates signal flow direction.

The graphics editor has two operation modes, called *symbol editor* (for edit of a symbol) and *contents editor* (for edit of a contents).
All symbol elements and contents objects are defined on the same coordinate system. This coordinate system has an upper left hand origin (same origin as the GPR coordinate system). However, this origin can be altered easily.
The coordinate system is shown as grid of horizontal and vertical lines.

2.1 SYMBOL EDITOR

All symbol elements reside inside the bounding box of the symbol. This bounding box has the dimensions of the contents bounding box which can't be changed with the symbol editor. Whenever the dimensions of the contents bounding box are changed, the symbol bounding box is adjusted accordingly. Symbol elements extending beyond the new bounding box are deleted.
Symbol elements are defined on grid-coordinates: both the starting point and ending point of a line are on grid-coordinates, as well as the centre of a circle, whereas the circle radius is a multiple of the grid-size.
Symbol coordinates are realtive to the upper left hand coordinate of the symbol bounding box.
No viewing operations such as zooming in and out are allowed. The only viewing operation implemented is setting the grid on and off. Future extensions could include a denser grid for creation of a more detailed symbol and viewing operations such as zooming.

The symbol editor has a set of manipulation commands to operate on symbol elements. This set currently includes only adding and deleting of elements.

2.2 CONTENTS EDITOR

With the contents editor the objects of a template contents can be edited. All objects are defined on grid coordinates. Both ending points of a wire are defined on grid coordinates. Instance origin (instance upper left hand corner) is defined on a grid coordinate as well. Instance width and height are multiples of the grid size. The contents editor has five sets of commands:

manipulation set includes the following types of commands:

- *add commands* for adding instances, terminals and wires.
- *delete commands* for deleting instances, terminals and wires.
- *move commands* for moving instances and terminals.
- *copy commands* for copying instances and terminals.
- *rotate command* for rotating instances.
- *mirror commands* for mirroring instances.

viewing set this set includes commands that affect the appearance of the displayed circuit (e.g. zooming in and out and selecting a new view-centre).

information set this set includes commands to retreive contents information. The information is shown in a popped up text-area.

help set this set includes commands to show help information as a user's guide. The provided help commands are in a preliminary stadium, but can easily be changed into definite versions.

miscellaneous set this set includes other commands like changing the current template, saving the template information, changing editor mode and leaving the editor.

2.3 GRAPHICS APPLICATION IN A WINDOWING SYSTEM

On APOLLO workstations windows are controlled by a program called the Display Manager. The Display Manager supervises the creation of computing environments (processes) in which programs are executed [5].
The graphics application repeatedly acquires and releases the display for brief periods of graphics operations. This gives the advantage of speed of operation while preserving the Display Manager's control over display functions such as changing window size and moving windows. The window in which the graphics application is executing can be manipulated just like any other window:

* the window size can be enlarged or reduced.
* the window can be moved to another position on the screen.

* the window can be pushed (moved to the bottom of the window stack thus obscuring the window) and popped (put on top of the window stack thus making the window visible). Default the contents of the window is shown in the visible parts of the window. However, this can be overridden by setting the no-refresh option.

2.4 BITPLANES

The editor program runs on APOLLO workstations having a display with at least eight bitplanes. The program makes use of the following bitplanes (bitplane 3 is used by the Display Manager for display of window borders and window backgrounds):

bitplane 0	white, used for cursor display, rubberbanding and drawing text.
bitplane 1	magenta, used for drawing instances and symbols.
bitplane 2	blue, used for drawing terminals.
bitplane 4	yellow, used for drawing wires.
bitplane 5	black, used for drawing the text background.
bitplane 6	red, used for highlighting menu items.
bitplane 7	green, used to display the grid.

Defining a plane for each contents object gives the advantage of adding and deleting of objects, without redraw of the total window. Futhermore, displaying and deleting pop-up menus runs very fast.

3. *EDITOR MODULES*

The software of the editor program is modular structured as shown in Figure 3. The top level module (*main module*) controls the middle level modules (*text input manager ... external database manager*), which themselves control the bottom level modules (*drawing module* and *mouse input module*). The arrows in the Figure indicate data flow direction. The high level interface between main module and middle level modules has only such parameters as world coordinates and element pointers. World coordinates are either absolute contents coordinates or relative symbol coordinates. An element is either a contents object (instance, terminal or wire) or a symbol element (line or circle).
The low level interface between the middle level modules and the bottom level modules has bitmap position as a parameter.

Command handlers are grouped in sets of related command handlers. All command handlers from the same set have nearly the same format. The contents editor has five sets of command handlers:

instance command handling set
includes command handlers related to instances only. The following commands are handled:

— *all commands from the instance menu.*
— *add instance command (main menu).*

terminal command handling set
commands related to terminals only are included. Handles the following commands:

— *all commands from the terminal menu.*
— *add terminal command (main menu).*

wire command handling set
includes command handlers related to wires only. Handles the following commands:

— *all commands from the wire menu.*
— *add wire command (main menu).*

view command handling set
includes command handlers to alter the view. Handles all commands from the view menu.

help command handling set
include command handlers for the following commands:

— *help main*
gives general help information on the contents editor.
— *help instance*
gives general help information on instances.
— *help terminal*

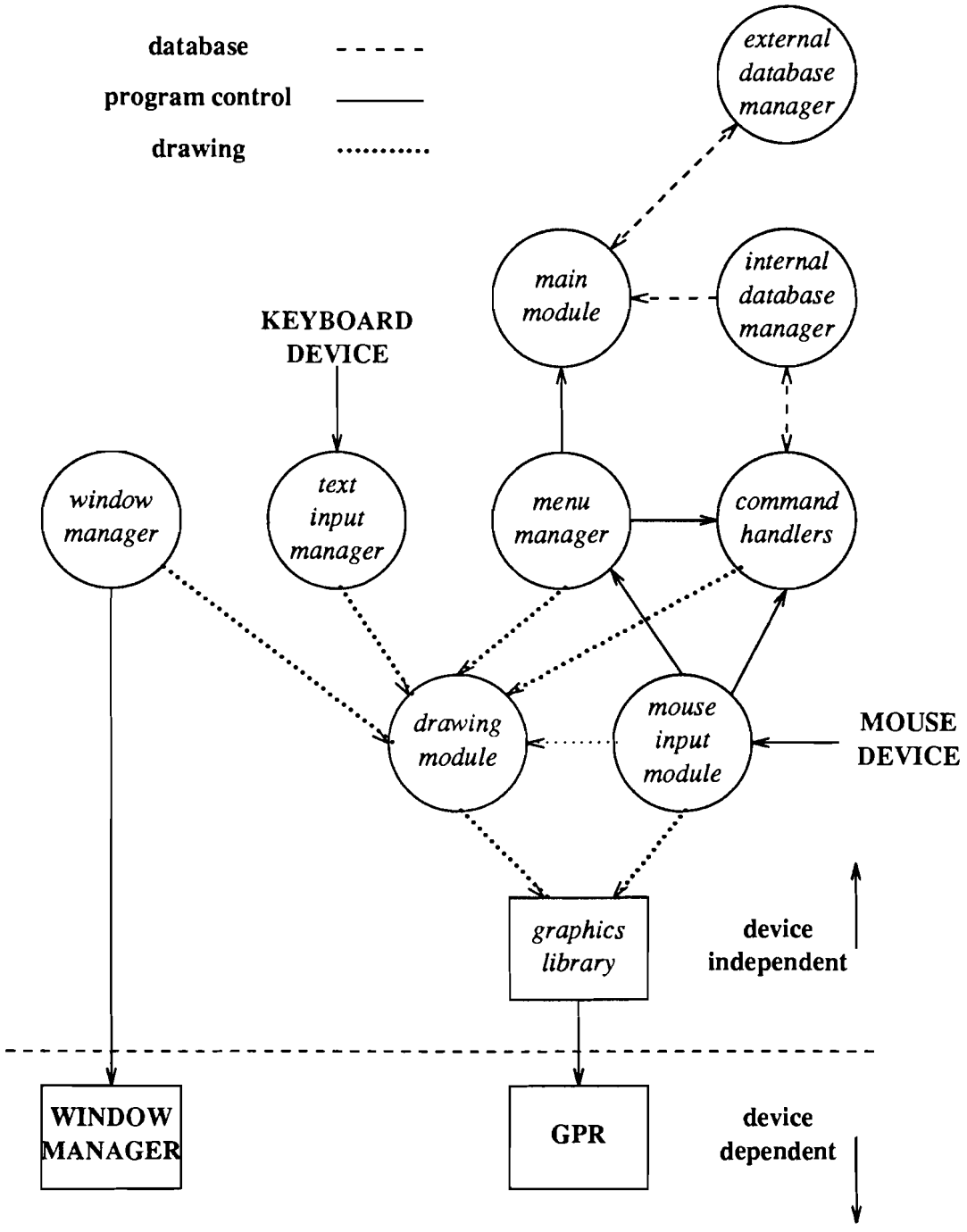

Figure 1. modular structure

gives general help information on terminals.
— *help wire*
gives general help information on wires.

The symbol editor has three sets of command handlers:

line command handling set
includes command handlers for the following commands:

— *add line*
— *delete line*

circle command handling set
includes command handlers for the following commands:

— *add circle*
— *delete circle*

view command handling set
includes just one command handler:

— *toggle grid*

3.1 MAIN MODULE

The main module controls the creation and maintenance of the template files. Furthermore, it receives mouse input events (button events only) to synchronize program execution. The internal database of the current template is searched to determine (together with the button event) which command handler is to be selected. If, for instance, in the contents area the repeat button is pressed at a world position occupied by a terminal, the command handler for the last selected terminal command is selected.
The algorithmic description of the main module (contents part), expressed in a pseudo C language, is stated below, see figure 2. The symbol part has a similar algorithm except for the help button case. In the algorithm described in this figure *contents object* is one of *instance*, *terminal* or *wire*. When replacing *contents object* by *symbol element* and deleting the help button case, the algorithm for the symbol part is obtained (*symbol element* is one of *line* or *circle*).

3.2 WINDOW MANAGER

The window manager module is a rather simple one, because the Display Manager takes over display functions such as moving the window, growing the window (enlarging or reducing window size), pushing the window and popping the window.
This module contains routines to create and delete windows.
Furthermore, the window manager module provides an editor window refresh routine. The refresh routine is called whenever the appearance of the editor window is affected. This routine is called by the GPR software upon interrupt received from the Display Manager.
At startup, a no-refresh option can be given. With this option the editor window is not refreshed

```
while (true)
 {
  wait for mouse button event
  convert bitmap position to contents coordinate
  switch (mouse button)
   {
    case select button:
     search internal database for a contents object
     if (object found)
        pop up the object menu and select an object command
        call command handler for the selected command
     else
        pop up main menu and select main command
        call command handler for selected command
     break;

    case repeat button:
     search internal database for a contents object
     if (object found)
        call command handler for the selected command
     else
        call command handler for selected command
     break;

    case help button:
     search internal database for a contents object
     if (object found)
        call object help handler
     else
        call main help handler
     break;
     }
 }
```

Figure 2. main module algorithm

when (part of) the window is obscured.

3.3 TEXT INPUT MANAGER

The text input manager module is the interface between editor program and the keyboard. The function of this module is creating input strings by allowing the user to type keyboard characters. The text input manager module is called whenever an input string is asked for.The maximum

string length is given as an argument.
The typed characters are made visible in a text area popped up at the current cursor position. A text-cursor indicates the position of the next character to be typed. The text-cursor disappears when the string has reached its maximum length.
The following set of characters is enabled: {"*a..z*", "*A..Z*", "*0..9*", "_", "/", *<return>*, *<backspace>*, *<F9>*}.
Aborting text input can be done by either typing key *<F9>* or by leaving the editor window.
The text input manager is called from the main module in three situations only:

— when no template argument (format: *-t<template_name>*) is given at startup.

— after selection of the **change template** command.

— after selection of the **add instance** command.

3.4 EXTERNAL DATABASE MANAGER

The external database manager module provides the means to save template information on disk and retreive template information from disk.
With each template a *creation date* and *last access date* is stored.
This module is called from the main module only.
The interface routines of this module take a pointer to a template as argument. Templates written to disk are saved in files having binary format. This gives two advantages:

— the internal database can easily be retreived because information is stored in a forward way.

— small amount of disk occupation.

If the information of the template, which is to become next editor object, is not stored internally it is read from disk. If the template file exists, the template datastructure is built from this file, whereas a new datastructure is created for non-existing template files.
When saving template information on disk, the template datastructure is maintained in internal memory.

3.5 INTERNAL DATABASE MANAGER

The representation of the internal datastructure is shielded from the other modules by internal database interface routines. Only inside the database manager modules (both internal and external) the appearance of the template datastructure is known.
The interface routines include a set of inquiry routines for retreiving internal database information. These inquiry routines are called from the main module and the command handlers.
Furthermore, for each contents object and symbol element a set of manipulation routines is included. The manipulation routines are called from command handlers only. They have similar names to the command handlers they are called from. For instance, the terminal command handler *copy_terminal* calls the manipulation routine *cpy_terminal* when copying a terminal.
All the internal database interface routines take a pointer to a contents object or a symbol element as argument.

The datastructure of all template files accessed during an edit session are permanently stored in main memory. The datastructure of instanced templates (in current templates only) are read into main memory also. This leads to the advantage of fast database information access. On the other hand it could mean a disadvantage while consuming large amounts of memory (especially when many large templates are edited). It is worth mentioning that this trade-off between access time and memory consumption hasn't been investigated yet.
An instance has a pointer to the invocated template, thus in the datastructure only instance origin, rotation and dimensions (for ease of internal database search routine) are stored.
Each time a new contents object is added to the template, the template bounds are adjusted. The exact template bounds are computed when the symbol bounding box needs to be drawn (either in symbol mode or when instancing a template).

3.6 MENU MANAGER

The menu manager is an attractive feature of the editor. It provides fast command selection in a user-friendly way.
This module is called from the main module and the command handlers.
When a command needs to be selected a menu is popped up at the current cursor position. Menus can have up to fifteen items, however, this number can easily be increased by change of a constant. Maximum item length is restricted to fifteen characters (can be increased by change of constant). All item strings of a menu are placed in a character-array. Length of this array is the number of items, width is the maximum item length. A pointer to this array is given as a parameter to the menu manager. By changing this array (size and/or strings) menus can be altered.
The font used to write the menu text to screen is called the menu font. By changing the font name (a constant) a new menu font can be chosen.
The number of items, together with the menu font used and string length of the biggest string in the character array, are used to compute the area size the menu occupies. The items are placed beneath each other. The first item is placed on top, the second underneath and so on.
When popped up, a default item is highlighted. To highlight another item the mouse is moved up or down. An item is selected by the mouse button transition given as an argument by the called module. This button transition is either select button down (pressing the middle mouse button, which is the select button) or select button up (release of select button). When the mouse is moved out of the menu area or when an item is selected, the menu is deleted. The number returned to the calling module is zero if no item was selected, one if the top item is selected, two for the next item and so on.

3.7 DRAWING MODULE

The drawing module is the interface between the middle level modules and the device-independent graphics library. It contains two sets of drawing routines, a set of inquiry routines and a set of control routines.
The high level drawing routine set includes drawing routines for drawing contents objects and symbol elements, routines for rubberbanding wires, lines and circles and routines for dragging terminals and instance boxes. These routines are called from the window manager module and the command handlers. The routines for drawing contents objects (symbol elements) take a pointer to the object (element) as argument.
The high level drawing routines are converted to the low level drawing routines. The low level

drawing routine set includes routines for drawing text and graphic primitives such as lines, circles, triangles and rectangles. These low level drawing routines are called from the text input manager module and the menu manager module.
The inquiry routine set includes routines to retreive graphic information such as current bitmap dimensions, colour table setting, font attributes (size, text length), visible areas and more.
The control routine set includes routines to set the colour table, select a font, set a clipping area and more.

3.8 MOUSE INPUT MODULE

The mouse input module is the interface between editor program and the mouse device. The editor program is synchronized around input events received from the mouse device (and occasionally the keyboard device). These input events include mouse movement, mouse button transition and editor window transition (and occasionally keyboard input). An event occurs when input from an enabled event is generated.
The mouse input module includes inquiry routines to inquire about cursor position, event type (mouse movement, button transition, window transition,), event data and more.

3.9 GRAPHICS LIBRARY

The graphics library contains device-independent graphic routines. The purpose of the library is to shield the editor program from device-dependent graphics packages.
The graphic routines included in this library are mapped on GPR routines. GPR (Graphic Primitives Resource) is a graphics software package supplied on APOLLO workstations. The routines that make up the GPR library can be used to manipulate the least divisible graphic elements such as drawing operations, text fonts, pixel values and bitmaps. Only a subset of these GPR routines is used to build the device-independent graphics library. The GPR package provides a set of input routines that enable application programs to accept input from various input devices. If an event is enabled these input routines report each event of the enabled type to the program with event data (mouse button transition or keyboard input) and a cursor position.

All routines of the drawing module and most of the routines in the mouse input module are built upon GPR routines.

4. INTERNAL DATABASE

All database information concerning contents objects and symbol elements is ordered for ease of search. Therefore, each contents object (instance, terminal or wire) and symbol element (line or circle) has its own structure defined. All object and element structures are orderly stored in double-linked lists. The template structure has pointers to all these lists.

4.1 CONTENTS OBJECT STORAGE

The coordinate-system of the contents has an upper left-hand origin. Terminal structures, containing terminal information, are stored in the terminal list, sorted on increasing x-coordinate. Terminals having the same x-coordinate are stored in sublists sorted on increasing y-coordinate.

Wire structures are stored in either the horizontal or vertical wire list. Wires stored in these lists, are sorted on their start position. For horizontal wires, the start position is the wire position with the smallest x-coordinate. For vertical wires, the start position is the position with the smallest y-coordinate.
Horizontal wires are stored in the horizontal wire list, sorted on increasing y-coordinate of their start position. Horizontal wires having the same y-coordinate are stored in sublists sorted on increasing x-coordinate.
Vertical wires are stored in the vertical wire list, sorted on increasing x-coordinate. Vertical wires having the same x-coordinate are stored in sublists sorted on increasing y-coordinate.

Instance structures are stored in the instance list. This list is sorted on increasing x-coordinate of the instance origin. Instance origin is the upper left hand corner of the instance bounding box. Instances having the same x-coordinate are stored in the same list, sorted on increasing y-coordinate.
All contents objects have absolute coordinates.

4.2 SYMBOL ELEMENT STORAGE

The coordinate-system of the symbol has an upper left-hand origin. Symbol lines are not restricted to any direction: they can have any direction. Line structures are stored in the line list, sorted on increasing x-coordinate of their upper left hand end point. Lines having the same x-coordinate are stored in the same list, sorted on increasing y-coordinate.

Circle structures are stored in the circle list, sorted on increasing x-coordinate of their centre. Circles having the same x-coordinate are stored in the same list, sorted on increasing y-coordinate.

Line end points and circle centre are relative to the upper left hand corner of the symbol bounding box. In this way, rotation and mirroring of instances can be executed very easy.

5. GMR APPLICATION

As a sidestep, some time was spent on the development of an alternative graphics editor, based on an additional graphics package supplied on APOLLO workstations. This so called 2D GMR package has its own internal datastructure, optimized for graphics operations.
Because this alternative editor is not to be used, it is only briefly described. Some conclusions resulted from this alternative research are stated at the end of this chapter. The 2D GMR package (*two-dimensional graphics metafile resource* package) is a collection of routines that provide the ability to create, display, edit and store device-independent files of picture data.
The standard form of data storage is a *metafile*. Metafiles contain *segments*. Each segment is a named entity consisting of a sequence of commands, used to build a graphic image. Commands describe the least divisible components of the picture.

The 2D GMR routines are categorized into *modeling* routines and *viewing* routines:

modeling routines	control and edit metafiles.
viewing routines	affect the form in which picture data within metafiles is displayed.

The 2D GMR package does not operate directly on bitmaps. Instead, the 2D GMR routines modify either the contents of a metafile or the manner in which the metafile is displayed.

5.1 EDITOR PROGRAM

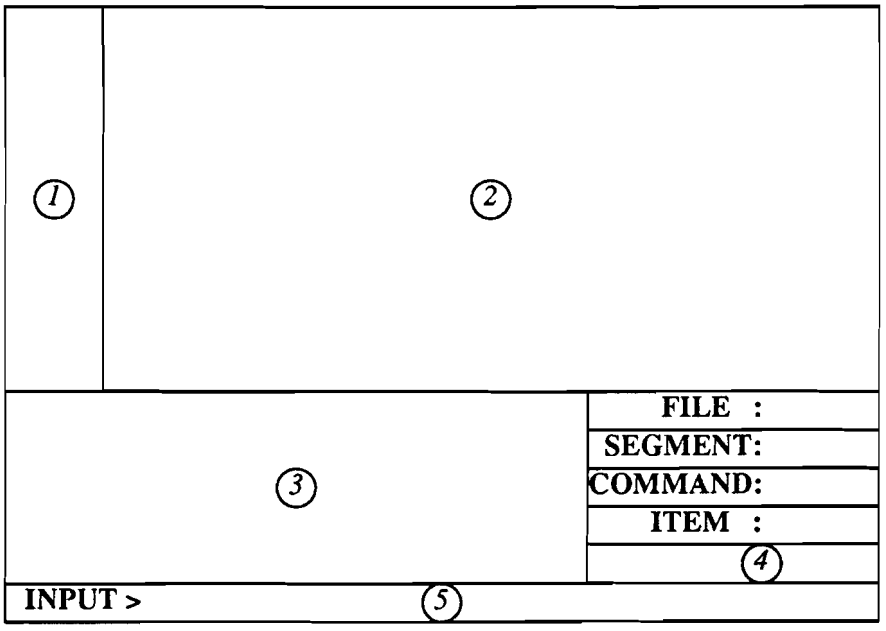

Figure 3. program window

The editor, created with the 2D GMR package, runs in its own window. This window is divided into five areas (see Figure 3) :

1 **menu area**	in this area either the *command menu* or the *item menu* is displayed. The *command menu* is used for selection of a command. The *item menu* is

	used for additional input to a command. N.B. a command is not an element of a metafile, but an editor action.
2 **work area**	in this area the current segment (or part of it) is displayed.
3 **text area**	this area is used by the program to write lines of help information.
4 **status area**	in this area *current file*, *current segment*, *current (editor) command* and *current item* are shown.
5 **input line**	used for text input.

All editor commands are briefly described below.

next segment	the user is prompted for a segment name. The named segment becomes current segment and is displayed in the work area.
next file	the current file is saved and closed. The user is prompted to give two names. First, a file name is given. The named file becomes current file. Then, a segment name is given. The named segment becomes current segment.
instance	the user is prompted for an segment name. The named segment can be instanced in the current segment.
zoom in	a zoom-in area can be defined by rubberbanding a rectangle. The view is zoomed in at the defined area.
add	a *line*, *rectangle*, *circle*, *instance* or *text* can be added to the current segment. Rubberbanding is provided.
move	for moving the above mentioned items. Dragging is provided.
delete	a selected item is deleted.
quit	exit from program.

A help button is provided to show help information in the text area.
Furthermore, some keys can be used to execute viewing operations, such as zoom in, zoom-out, translate the view and reset the view.

5.2 MODULAR STRUCTURE

The program is modular structured, as shown in Figure 4. The *command control module* actually consists of five independent modules, as shown in Figure 5. All five modules are controlled by the main module. Each command can have up to five states:

initial state
in this state the command is initalized. For instance, initialisation of the *add* comand means setting up the *item* menu in th menu area.

start state
when the execution button is pressed in the work area, the command is started. For instance, ruberbanding mode can be set.

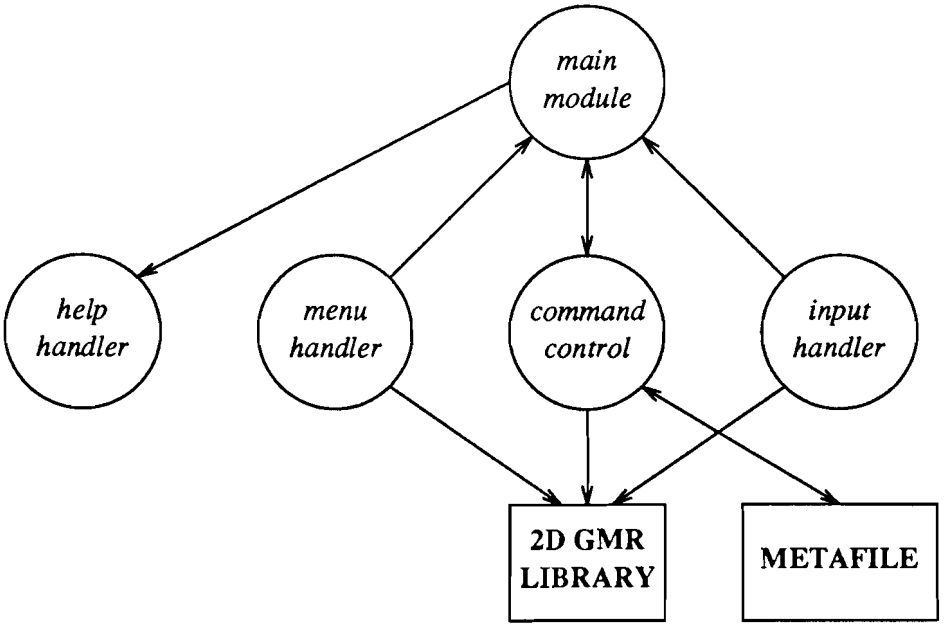

Figure 4. modular structure

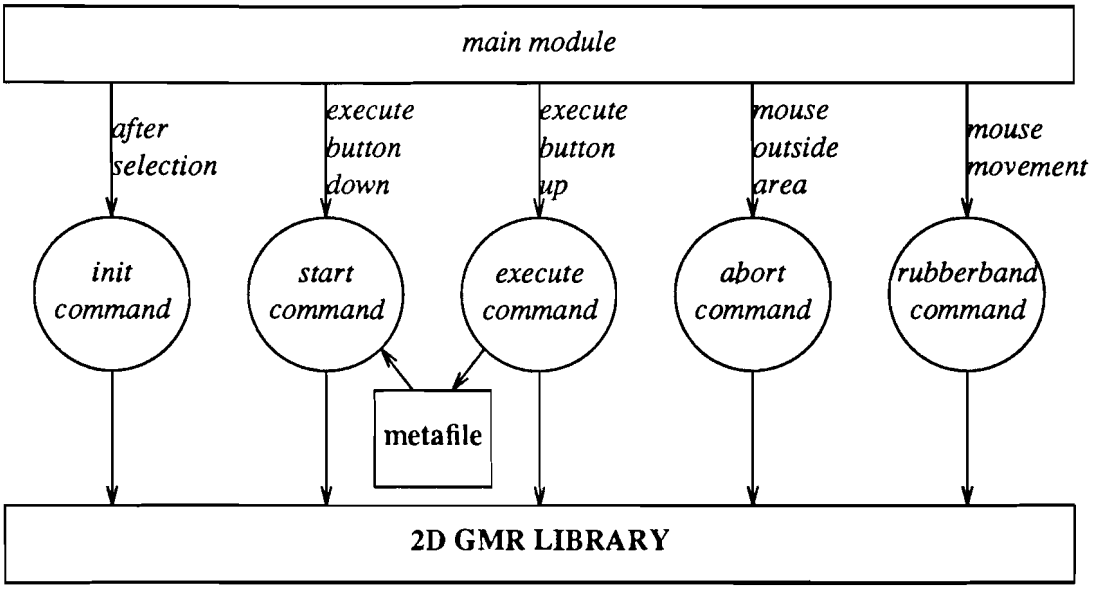

Figure 5. command control

execution state

when the execution button is released inside the work area, the command is executed.

abort state

execution of the command is aborted when the cursor leaves the work area.

rubberband state

the execution button is held down and the mouse is moved. For instance, when

adding a rectangle the cursor position is the corner of a rubberbanded rectangle.

5.3 CONCLUSIONS ON GMR

5.3.1 Advantages

metafile
internal and external database modules, for storing picture data in main memory and on disk, are provided by the package.

viewing operations
viewing operations, such as zooming (in and out), translating are provided by the package.

rubberbanding
the package also provides rubberbanding.

instance operations
instance scaling, translation and rotation operations are included in the package.

5.3.2 Disadvantages

metafile
all picture data is stored in commands and segments. This makes it very hard to correlate the graphic elements to network database information.

portability
because the package has such a huge impact on the program, it will be very difficult to create interface routines for the package.

not user-friendly
the package is difficult to program with. This is partly due to incomplete documentation of the package.

6. CONCLUSIONS

The editor program has a very flexible structure. Parts of the program can be changed without affecting the rest of the program:

— *internal datastructure*

— *external datastructure*

— *menus*

— *command handlers*

— *help facility*

The program is user-friendly:

— it provides pop-up menus with clear indication of current item.

— text input is given at the place it is asked for.

— repeat button for fast repeat of a command.

— many view commands for display of another part of the circuit.

— help facility to guide non-experienced users.

— rubberbanding of objects and elements.

— the program runs in a multitasking environment.

— the program window can be handled just like any other window.

Device dependency is restricted to the graphics library module only.

Hardware requirements:

— bitmapped display with eight planes and a high resolution.

— user definable colourtable.

— mouse device.

Software requirements:

— graphics package with mouse support, clipping and colourtable setting.

7. *RECOMMENDATIONS*

A lot of small recommendations can be made to extend the editor program. All of these recommendations can easily be implemented. They are briefly described below.

instance scaling

this is by far the most important recommendation. With the bottom up design strategy the template size is increased with each higher level. When template size increases, the grid size decreases. With a small grid size it is difficult to add new wires and terminals to the contents. When the instance size is scaled down, template dimensions can be kept small.

instance disclosure

sometimes called instance explosion.

With this feature the contents of the invocated template be made visible inside the instance box. Disclosure of an instance can be turned of by *instance enfolding*. With instance enfolding, the symbol of the invocated template is displayed inside the instance box.

This disclosure/enfolding feature was reckoned with when defining the template symbol bounding box to have dimensions equal to the template contents bounding box dimensions. Implementing the disclosure feature can be made much easier when defining all contents objects relative to the upper left hand corner of the the template bounding box.

level down command

this command can be added to the instance menu. Selection of this command makes the invocated template current editor object. This command is especially useful when following the top down design strategy. The previous editor object is put on top of a template stack. The template on top of the stack can be made current editor object when issueing the level up command (see below).

level up command

this command can be added to the main menu. Selection of this command pops the template stack: the template on top of the template stack becomes editor object.

extension of help facility

the currently implemented help facility can be regarded as a frame for a really helpful editor feature.

extension of main contents menu

the following commands can be added to the main menu in contents mode:

— *copy area command:*
just like with the **zoom in** command, an area can be defined by rubberbanding a rectangle. Having defined the area all contents objects inside this area can be copied to another place in the contents area. An additional feature could be rotation and even mirroring of the selected area.

— *split area command:*
by selecting two grid points in the contents area a split vector is defined. All

contents objects to the right of the left split point are shifted to the right over the horizontal split distance. Horizontal wires extending beyond the left split point are lengthened with the horizontal split distance. A similar reasoning holds for the objects below the top split position: they are shifted downward over the vertical split distance.

extension of symbol editor

— define a *finer symbol grid* (for drawing symbol elements).

— implement more *viewing commands* to alter the symbol view (e.g. zoom in and out, shift, centre).

— define *more symbol elements* such as arcs and text.

draw unique instance name

inside the instance box a unique instance name (and/or the name of the invocated template) can be drawn. These names are not drawn when the grid size becomes too small.

draw unique terminal name

see previous recommendation.

startup picture

when no template aregument is given at startup, a picture (containing the name of the program and some other information) is drawn in the window area.

undo handler

the most recent commands can be kept in an undo stack. By selecting the undo handler the command on top of this stack is undone and deleted from the stack.

8. REFERENCES

[1] Lodder A., Stiphout M.T. van, Eijndhoven J.T.J van
ESCHER: Eindhoven SCHematic EditoR reference manual
Eindhoven University of Technology, Department of Electrical Engineering,
EUT report 86-E-157, 1986 Eindhoven, The Netherlands

[2] van Stiphout M.T. van
Design and implementation of a schematics entry program
Eindhoven University of Technology, Department of Electrical Engineering,
Master thesis, 1986 Eindhoven, The Netherlands

[3] Wilde P. de, (editor)
The Integrated Circuit Design book
Delft University Press, 1986 Delft, The Netherlands

[4] Steen H.L.J. van der
Interactive event-driven simulation
Eindhoven University of Technology, Department of Electrical Engineering,
Master thesis, 1986 Eindhoven, The Netherlands

[5] *Programming with DOMAIN Graphics Primitives*
Apollo manual, 1987

[6] *DOMAIN Graphics Primitive Resource Call Reference*
Apollo manual, 1987

[7] Fleurkens J.W.G.
HILDE
A high level design environment in CommonLisp
Eindhoven University of Technology, Department of Electrical Engineering,
Master thesis, 1988 Eindhoven, The Netherlands

9. USER MANUAL

This chapter could serve as an editor reference manual.
All commands, both in contents and symbol mode, are briefly described. As an illustration, some figures are added at the end of this chapter.

9.1 CONFIGURATION

On initialisation the editor opens a window, having default width of 700 pixels and default height of 400 pixels, at the upper left hand corner of the screen.
The default colour setting for the graphic objects and menus is:

- *white* for cursor, rubberbanding and info and help text.
- *green* for the grid (both in the symbol and contents mode).
- *yellow* for wires.
- *blue* for terminals.
- *magenta* for instances and symbols.
- *black* for text area background (for *menu, info* and *help* text).
- *red* for highlighting menu items.

Both the window size and colour setting can be changed at startup by using appropriate arguments. With the *-c<file name>* option a *configuration file* having the following format can be given at startup:

<window name>
<window width> <window height>
<grid size>

With the *-g<file name>* option a *colour file* having the following format can be given to set the colours:

rubberband	*<red index> <green index> <blue index>*
highlight	*<red index> <green index> <blue index>*
text_background	*<red index> <green index> <blue index>*
wire	*<red index> <green index> <blue index>*
terminal	*<red index> <green index> <blue index>*
instance	*<red index> <green index> <blue index>*
grid	*<red index> <green index> <blue index>*

The order in which these lines appear is of no interest.

A so called *no-refresh option* can be given as an argument at startup (format *-r*). With this option the editor window is not refreshed when (part of) the window is obscured. The visible parts of the obscured editor window are shown in a window background color provided by the Display Manager.
When this option is not given, the visible parts of the editor window are redrawn every time the

appearance of the window is affected. With the window obscured, the instances are shown only by their bounding box (for speed up of window refresh).

The window in which the editor is executing has no areas for static display of menus and status. All of the area is used for display of templates. The window area gives a view of either the template contents or the template symbol. In contents mode this view can be altered with commands from the viewing set. In symbol mode the view can not be altered.

9.2 INPUT DEVICES

Current cursor position is properly indicated by a crosshair cursor, which is tracked by movement of the mouse. The whole editor program is built around input events received from the mouse device. The mouse device has three buttons, all of which are used by the program. The left button serves as repeat button. A repeat button gives the advantage of fast repeat of the last selected command, without going through the command selection again.
The middle mouse button is the select button. Its function is selection of menu items. A menu item can be either a command or a new menu. A menu item is selected either by a downward transition of the select button (that is, pressing the middle mouse button) or by an upward transition of the select button (that is, release of the middle mouse button).
The right mouse button has a guidance purpose. When this button is pressed it gives either general or detailed editor information, both currently only in contents mode.
Both the left and right mouse button have an additional function. While executing a command (e.g. moving an instance or a terminal) pressing either button aborts the command.
Optionally text-input can be given by typing keyboard characters. A rather simple line editor is provided.
Not all characters can be used for building the string. When a character is disabled a beep is sound.

9.3 MENU DESCRIPTION

Command selection is done by selecting an item of a pop-up menu.
There are several types of pop-up menus: the contents editor has four menus (*main, instance, terminal* and *wire menu*), the symbol editor also has four menus (*main, edit, line* and *circle menu*).
By pressing the mouse-select-button a menu is popped up. Its type depends on current editor mode and current cursor position. In contents mode, if an object (*terminal, wire* or *instance*) is found at the current cursor position the corresponding menu is popped up. If no object is found the main menu is popped up.
In symbol mode the main menu is popped up when the cursor is outside the symbol bounding box. When inside this box, if a symbol element (either a *line* or a *circle*) is found the corresponding menu is popped up, whereas the edit menu is popped up if no element is found.
With a menu popped up, a default item is highlighted, indicating the item to be selected when the select button is released. This highlighted item is either the last selected item, if one, or an item chosen by the program itself (either the top item or the middle item).
By moving the mouse up or down, while holding the select button pressed down, a menu item is selected by release of the select button. After selection the menu is deleted. When the cursor is moved outside the menu area the menu is deleted leaving no item selected.
By pressing the repeat button on the mouse, the last selected command is repeated.

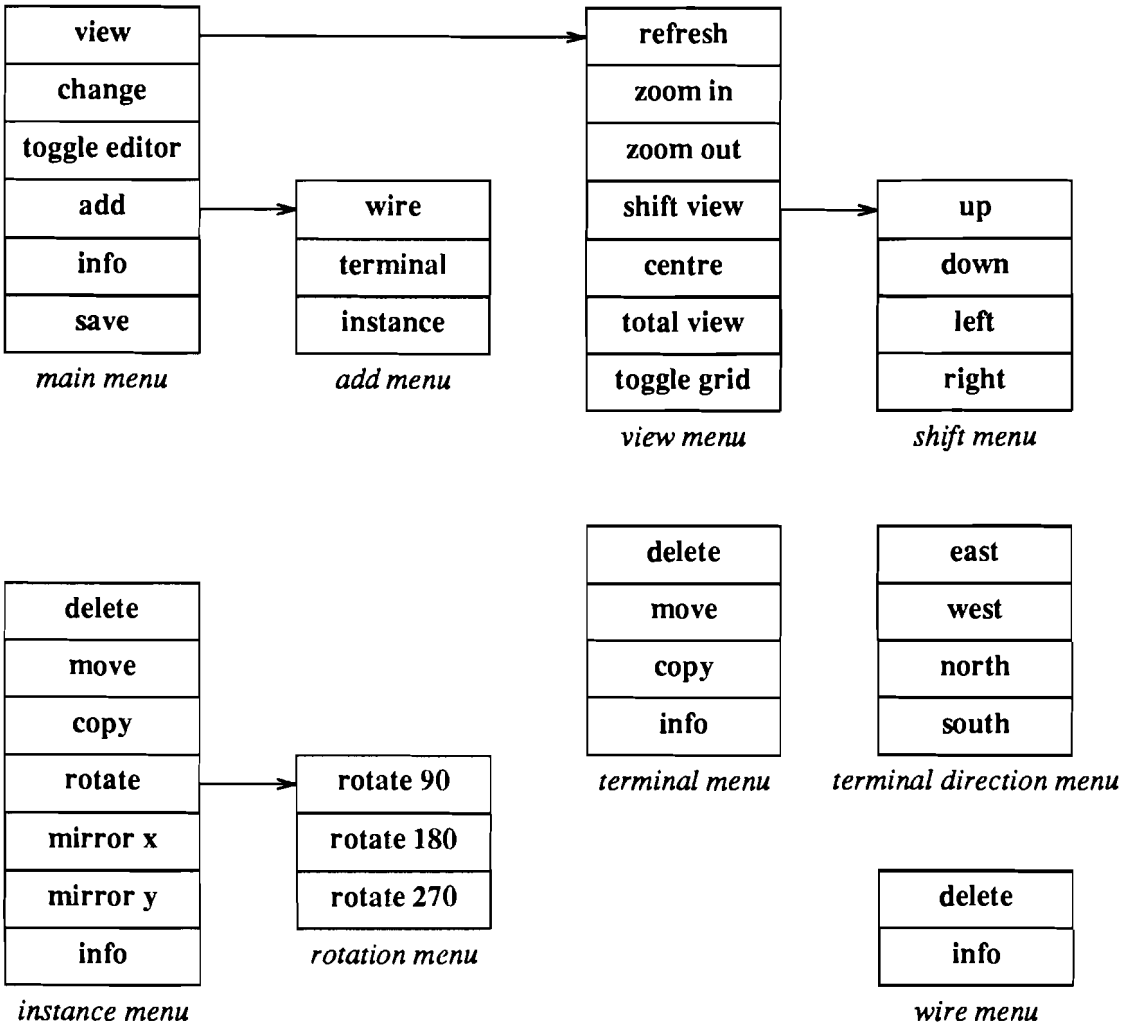

Figure 6. contents menus

By pressing the help button, help information is popped up at the current cursor position. The type of help information depends on the object found at the cursor position. The help information is deleted when the cursor is moved outside the text area. Currently, this help feature is implemented only in contents mode.
In Figure 1 all menus available in contents mode are shown. An arrow after a menu item indicates the next menu to be popped up when this item is selected.
In Figure 2 all menus available in symbol mode are shown.

9.4 COMMAND DESCRIPTION

In this section all editor commands are briefly described. It is divided into two parts. The first part deals with contents mode commands, in the second part the commands available in symbol mode are described.

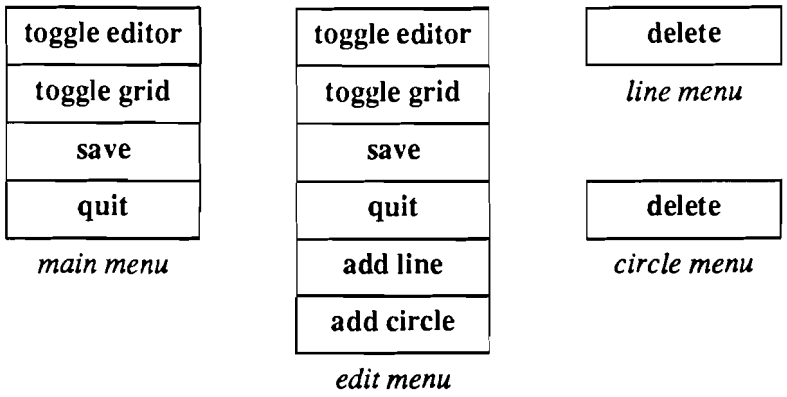

Figure 7. symbol menus

9.4.1 Contents Mode

The contents editor has four different sets of commands as described below. Edit commands can be used to edit the contents of the template. They include adding and deleting of all objects, whereas some objects can be moved, copied, rotated and mirrored. View commands affect the appearance of the view, that is the visible part of the contents. This visible part can be zoomed in and out and translated. A centre command is provided to select a new view centre. Furthermore, the grid can be set on and off. Info commands are are provided to give either general or detailed information on objects. Miscellaneous commands like changing current template, changing editor mode and program exit complete the set of contents commands.

9.4.1.1 edit commands

add instance the user is prompted for a template name. If the template exists the crosshair cursor disappears and the bounding box of the given template is displayed. The *origin* of this box (upper left hand corner) is located on the current cursor position. The origin of the box is tracked during movement of the mouse thus enabling dragging of the instance box in search for the right instance position. Dragging is done while holding the select button pressed down.
After having selected the instance position, the instance can be rotated by selecting an item of the rotation pop-up menu. Possible rotations are: *rotate over 90 degrees, rotate over 180 degrees* and *rotate over 270 degrees.* All rotation is clockwise, rotation origin is the origin of the instance.
After addition of an instance the symbol of the instanced template is displayed on the selected position.
By pressing the abort button while dragging the instance box, the command is aborted.

delete instance the selected instance is deleted.

move instance the bounding box of the selected instance can be dragged when holding the select button pressed down. After release of this button the instance is moved to the new position.
No cursor is displayed during dragging of the instance box.
The command is aborted when pressing the abort button while dragging the instance box.

copy instance same as move instance command except after release of the select button the instance is copied instead of moved.

rotate instance the rotation menu is popped up for selection of the rotation angle. Rotation is clockwise, rotation origin is the origin of the selected instance.

mirror instance an instance can be mirrored both in its central x-axis (*mirror_x* command) and central y-axis (*mirror_y* command).

add terminal the terminal direction menu menu is popped up for selection of the terminal direction.
The cursor is deleted and the terminal is dragged while holding down the select button and moving the mouse. Release of the select button adds the terminal to the contents, whereas by pressing the abort button, the command is aborted.

delete terminal the selected terminal is deleted.

move terminal the selected terminal is dragged to its new position by holding down the select button and moving the mouse. Release of select button moves the terminal to the new position. Pressing the abort button aborts the command.

copy terminal same as the move terminal command except after release of the select button the terminal is copied instead of moved.

add wire a point is selected which becomes the start point of the first wire. While holding down the select button and moving the mouse, the current cursor position is rubberbanded to the first wire-position. Release of the select button results in addition of the new wire. After addition of a wire the current cursor position is rubberbanded to the last selected end point of the new wire, thus enabling a next wire to be added.
No cursor is displayed during rubberbanding. Pressing the abort button aborts the command.
N.B. Only vertical and horizontal wires are allowed.

delete wire the selected wire is deleted.

9.4.1.2 view commands

zoom in a small part of the view is enlarged (by increasing the grid size) to show detailed information.

With the first point selected, the current cursor position acts as the opposite corner of a rubberbanded rectangle, indicating the zoom in area. Moving the mouse, while holding the select button pressed down, enlarges or reduces the zoom in area. By release of select button, the view is zoomed in at the selected area. The midpoint of this area becomes centre of the new view.
During rubberbanding no cursor is displayed. The command is aborted by pressing the abort button.

zoom out the reverse operation of the zoom in command: more of the circuit is displayed by redraw of the contents with a reduced grid size.

shift view the view is translated to display another part of the circuit. The shift menu is popped up for selection of the shift direction. Translation can take place in four directions: *left, right, up* and *down*.
When translating vertically (either *up* or *down*) the view is moved over a fourth of the display height. Horizontal translation (either *left* or *right*) moves the view over a fourth of the display width.

centre a selected point becomes centre of the new view.

total view the complete circuit is shown.

toggle grid if the grid is visible, it will be deleted. If it is invisible, it will be displayed.

9.4.1.3 info commands

main info shows information of the current template: *name, designer, size, creation date* and *last access date*.

instance info *name of instanced template, name of designer of instanced template, instance time, size, position, rotation* and *mirroring* of the selected instance are shown.

terminal info *position, type* (currently always *input), direction, terminal identifier* (currently zero) and *net identifier* (currently zero) are shown.

wire info *start position, end position, length, direction* and *net identifier* (currently zero) are shown.

The info text is displayed in a text area with the current cursor position as its centre.
No cursor is displayed. The info text is deleted when the cursor is moved out of the text area, enabling the cursor to be displayed.

9.4.1.4 miscellaneous commands

change template the user is prompted to give the name of another template, which is to become editor object.

toggle editor	switch to the symbol editor. The symbol of the template is displayed, if one, otherwise the symbol surrounding box is displayed.
save	save the template information (both contents and symbol information) on disk in a file named *<template>.temp*.
quit	leave the editor, without saving the template.

9.4.2 Symbol Mode

The symbol editor has three different sets of commands. Edit commands allow symbol elements to be added or deleted. A view command is provided to set the grid on and off. The third set includes miscellaneous commands like changing editor mode, save template information and program exit.

9.4.2.1 edit commands

add line	by selecting the first line end point, the cursor is rubberbanded to this point while holding the select button pressed down. Release of select button results in addition of the line if the second end point is inside the symbol bounding box. No cursor is displayed during rubberbanding. Pressing the abort button aborts the command. Lines of all directions can be added.
delete line	the selected line is deleted.
add circle	first the centre of the circle is selected. Then by holding the select button pressed down and moving the mouse, a circle is rubberbanded for defining its radius. Release of select button results in addition of the circle if the complete circle is inside the symbol bounding box. No cursor is displayed during rubberbanding. Pressing the abort button aborts the command.
delete circle	the selected circle is deleted.

9.4.2.2 view commands

toggle grid	if the grid is visible, it will be displayed, otherwise it is deleted.

9.4.2.3 miscellaneous commands

toggle editor	switch to contents mode. A total view of the contents is given.
save	save the template information (both contents and symbol information) on disk in a file named *<template>.temp*.
quit	leave the editor without saving the template.

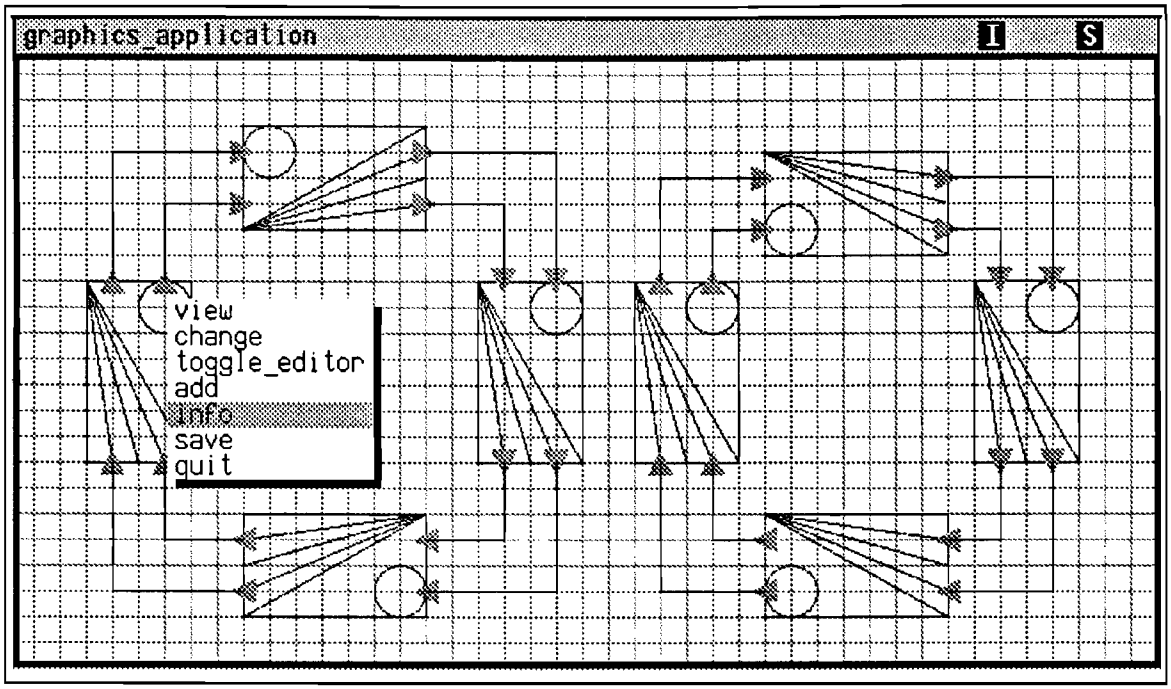

Figure 8. contents main menu

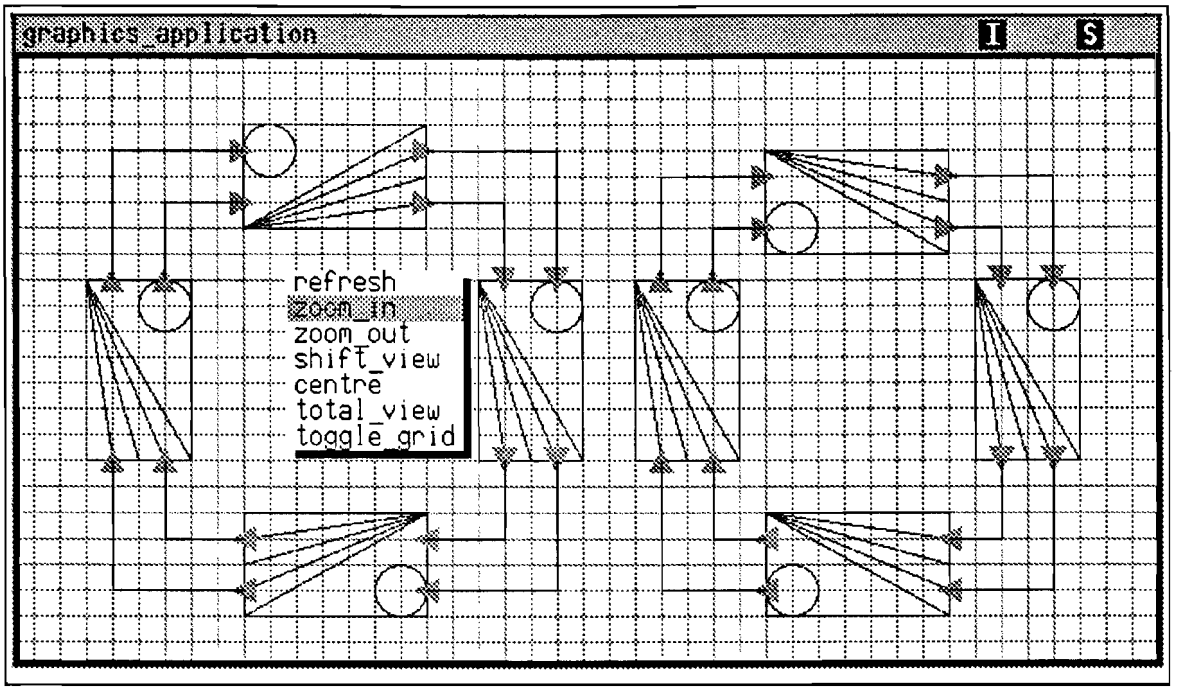

Figure 9. contents view menu

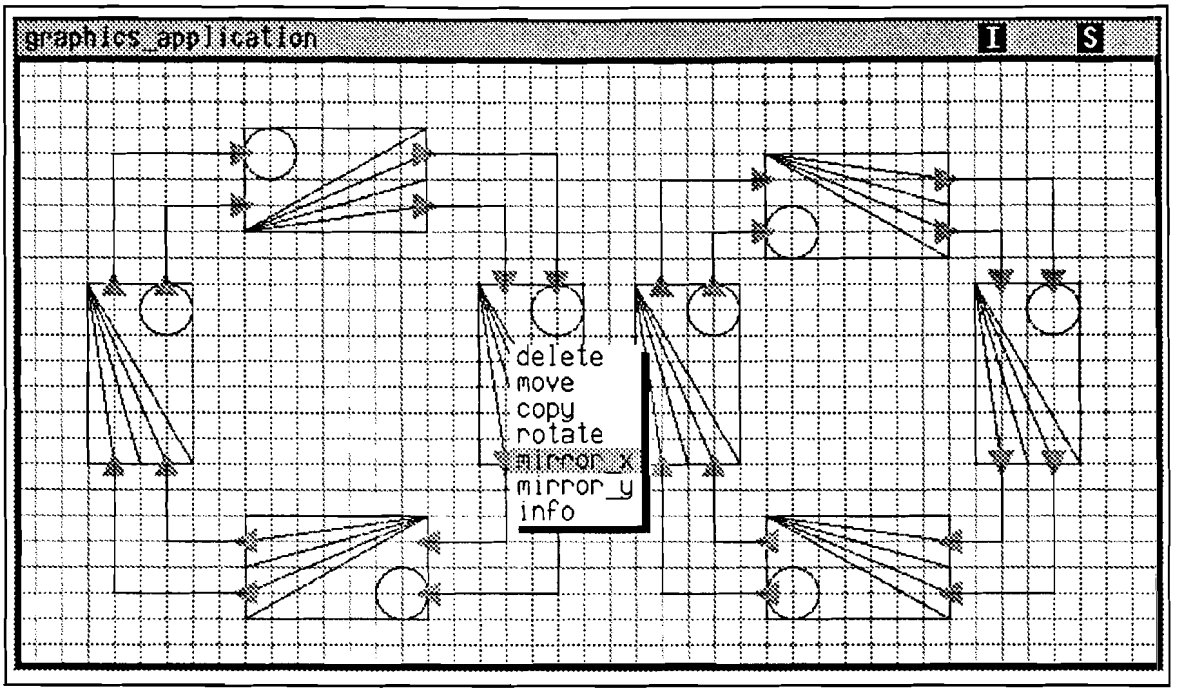

Figure 10. contents instance menu

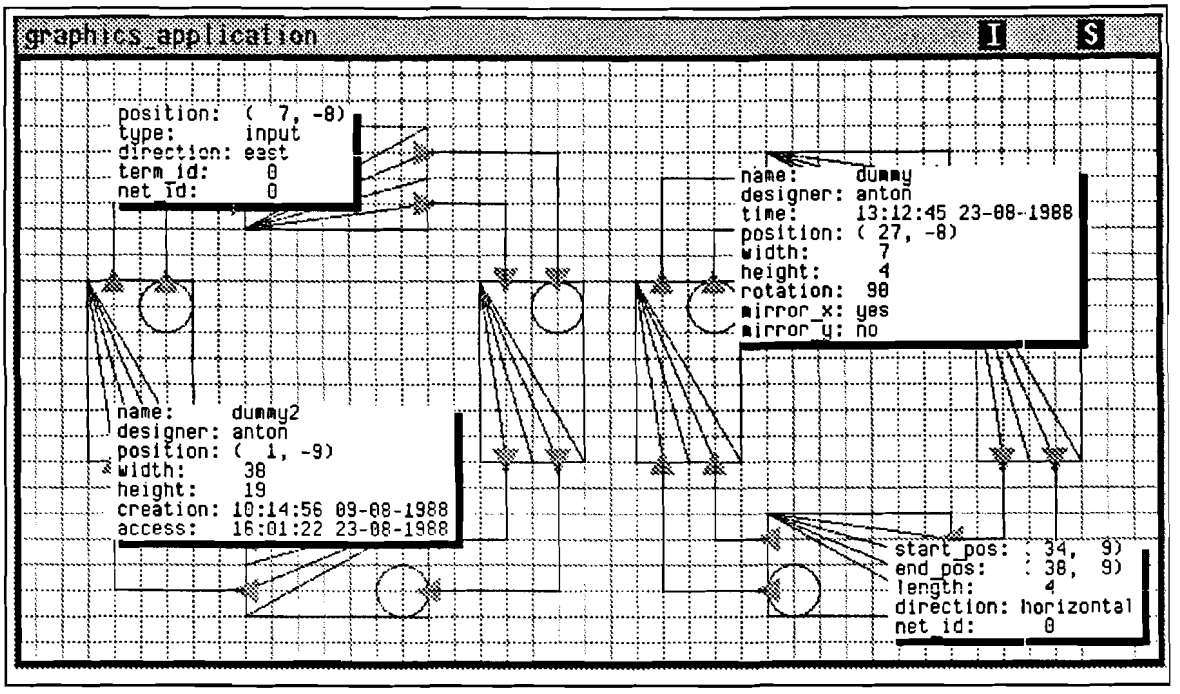

Figure 11. pop-up information