4

The Graphical Editor

4.1 Introduction

Today the specification of integrated circuits is based on two different methods: procedural or graphical. In the first case the specification is given in a hardware description language (HDL). In most cases these languages are derived from imperative or functional programming languages. They can be classified according to the represented properties of the circuit:

- ☐ languages that only describe the functional behaviour of the circuit without considering any structural properties, e. g. ISPS ([Bar78]) or Macpitts ([SSC82]).
- ☐ languages that describe both functional and structural properties, e. g. Zeus ([GL85]), Hades ([Wir82]), VHDL ([LSU89], [Sha82]) or EDIF ([Com87]).
- ☐ languages that only describe the structure of the circuit, e. g. HISDL ([Lim82]).

Common to all these languages is a great flexibility, because they allow the full power and generality of a programming language. Procedures are written to describe each part of the circuit and are combined hierarchically using the methods defined for programming languages. Parameters may be passed to procedures in order to generate different versions of the same structure.

But the procedural approach suffers from two drawbacks. First it is hard to embed graphical information in a textual procedure, in other words topological information is difficult to visualize when specified textually. Second the results of even small changes in the input cannot be seen without recompiling the whole circuit which can be a time–consuming process. Thus procedural systems tend not to be very interactive.

Systems which base on the graphical specification method offer a graphical editor to the user. This editor is used to specify structural information of a circuit, where subcircuits can be placed on the input screen and can be connected with lines. Some editors also allow the specification of behavioural aspects of a circuit. For instance in the system Gdl ([DBR$^+$88]) the user can draw petri nets in order to specify control and data flow of a circuit. In most cases however the graphical input is translated into interchange formats such that the knowledge of the designer given by the topological information will be lost.

Such editors normally build the frontends of commercial design systems. Although they support hierarchical specification they cannot be regarded as easy to use input tools. An important reason for this fact is that these commercial editors do not support the extraction of regular parts of circuits in order to set up parameterized and recursive circuit descriptions.

An exception to the rule is the layout system ESCHER ([CF86]) which allows parameterized circuit specifications. In this system the user can group cells together which are somewhat like one dimensional arrays in programming languages. The disadvantages of ESCHER follow from the fact that it is not based on a well defined mathematical calculus which leads to ambiguities in the interconnection of the above groups. Moreover ESCHER only supports the use of only one single parameter. This is an ugly constraint because a lot of interesting circuits can be easily described using more than one parameter. A well known example is the Wallace tree multiplier ([Wal64]) where a short recursive description with two parameters is given in [LV83]. A parameterized specification of this multiplier with the help of CADIC 's graphical editor will be introduced in section 10.2. ⇝ 10.2

4.2 Basic Concepts of CADIC

Because both approaches have their disadvantages we will now introduce a specification method which powerfully combines the flexibility of a programming language with an intuitive graphical representation. For this purpose the design level of CADIC is based on a mathematical calculus, which has been developed within the project B1 of the SFB124 and which serves as a programming language for the specification of integrated circuits. The properties of this calculus, which has originally been introduced in [Hot65] and [Hot74], are explicitly described in [Mol86]. In the following we will summarize the main aspects in order to understand the design level of CADIC .

The importance of boolean algebra in the field of the design of logic circuits is well known. It is the base of a method, which allows us to describe the logic of circuits and to exploit these objects for further calculations. This has been sufficient as long as the arrangement of the elementary gates and the connections between these did not play an important role. Now that technology has been improved and the integration of millions of transistors on a single chip can be achieved, the geometrical layout of the wires and the gates can no longer be neglected. In the design of very large circuits the geometrical layout directly influences the function of the circuit. From this point of view it is desirable to have a calculus, which combines the functional and geometrical aspects of a design.

In order to get an easy to handle method there will be taken some abstractions in a circuit description. Each circuit will be laid out into a rectangle, at the border of which its external connectors are placed. Within the rectangle there are components which are connected by wires. The whole layout is projected into the Euclidean plane, such that there are no different layers first. In order to receive a planar embedding, we consider branches and crossings of wires as basic components. Further we neglect the physical width of wires and look at them as sequences of line segments. Each component as well as the surrounding rectangle has a northern, eastern, southern and western border. It also has a name, and sub–circuits with the same names must have the same number and sequence of external connectors at each border. This abstract method of describing a circuit is called *logic topographical net.*

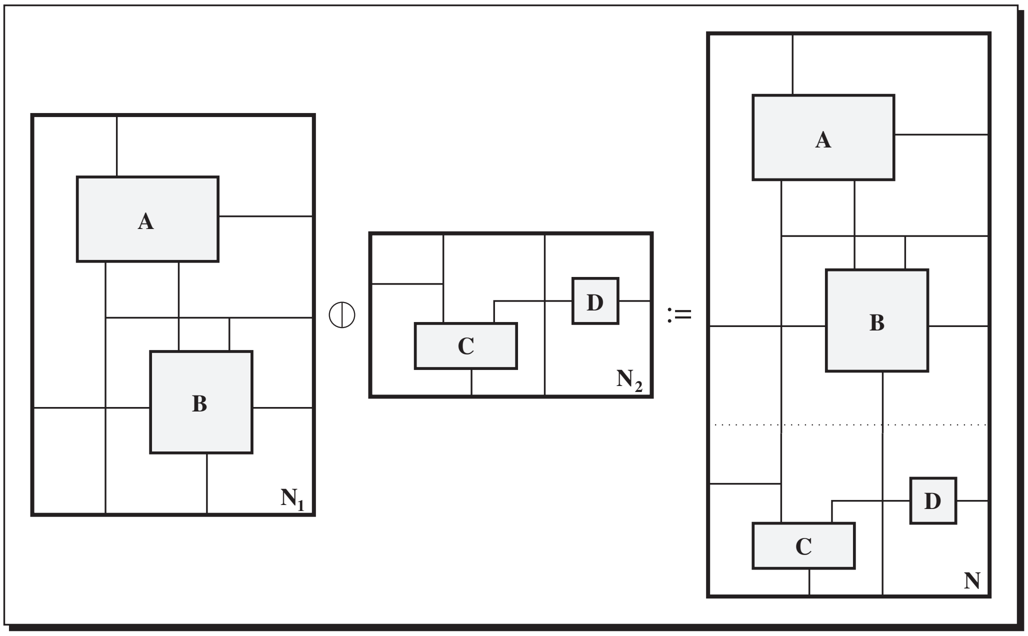


Figure 4.1: Vertical composition of two logic topographical nets

In order to construct larger nets we define two simple operations on logic topographical nets. Let N_1 and N_2 be two topographical nets. Then we define the vertical composition $N := N_1 \varobar N_2$, if and only if the southern border of N_1 and the northern border of N_2 fit together (c.f. figure 4.1).

In the same way we can define the horizontal composition $N := N_1 \ominus N_2$, if and only if the eastern border of N_1 and the western border of N_2 fit together (c.f. figure 4.2).

The resulting net N is given by the two parts N_1 and N_2, which are connected at their borders, i.e. the external connectors at these borders are identified and finally they are erased.

With the help of these operations we could create large logic topographical nets over the set of basic cells, branches, crossings and wire segments.

A disadvantage of this method is that we need an exact topographical de-

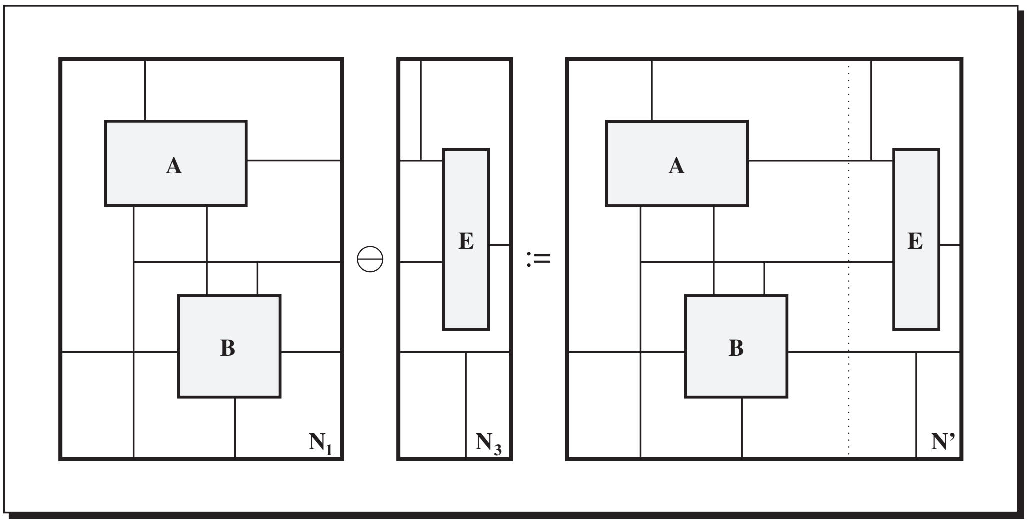


Figure 4.2: Horizontal composition of two logic topographical nets

scription of the resulting net. To overcome this restriction we introduce an equivalence relation between topographical nets. By that relation we consider two topographical nets to be equivalent, if they can be transformed into each another by a sequence of elementary deformations. Legal deformation steps are deformations of wires, moving, rotating and scaling of cells as well as repositioning external connectors at the borders of cells. A deformation is illegal, if for example the sequence of the connectors is changed of if additional crossings of wires are created. All resulting nets during a sequence of deformations must be legal topographical nets.

According to this set of elementary deformations the two logic topographical nets in figure 4.3 are equivalent. With this equivalence relation we obtain classes of logic topographical nets. Each class is called *logic topological net* and an element of a class is called a representative of a logic topological net. The vertical and horizontal composition can be applied to logic topological nets, if there exist two topographical representatives such that the operation is defined as shown above, i.e. if there exist two logic topographical nets, where the connectors can me moved to appropriate positions, such that the borders can be glued together.

In most cases the designer not only has to look at these numbers of con-

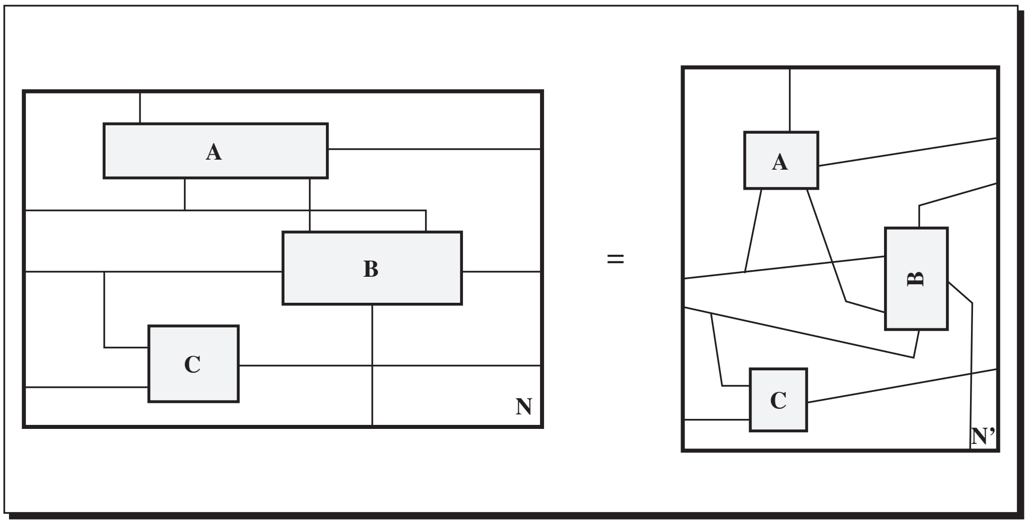


Figure 4.3: Two representatives of a logic topological net

nectors, but the connectors may have a specific type, such as clock signal or power supply signals. This fact is represented in the calculus by defining the composition operations, if the sequences of the signal types, read from left to right or top to bottom respectively, at the corresponding borders are identical.

The underlying mathematical calculus is called the *bicategory* of logic topological nets ([Mol86]). The description of a circuit within this calculus is done by *bicategorial expressions* which consist of the operations $\ominus$ and $\obar$ composing basic cells and wiring elements. The following is an example for a correct bicategorial expression, which we will call *net equation*, because it defines a *net variable* by the corresponding expression.

$$HalfAdder = (\vdash \ominus \neg \ominus 1_1^{\obar}) \obar (1_1^{\obar} \ominus \ulcorner \ominus + \ominus \dashv) \obar (AND \ominus EXOR)$$

In this example the symbol $\vdash$ stands for an east branch of a wire, $\neg$ denotes a south–west knee and *EXOR* is an element from the library of basic cells. The notation $1_1^{\obar}$ represents a vertival wire segment of width 1 which is the unit for the vertical composition.

This small example already indicates that bicategorial expressions can become very large and difficult to handle. To solve this problem we first introduce the hierarchy concept for bicategorial expressions.

In the sense of bicategories hierarchy means to transform a net over a set of cells A into a description over a set of cells B. This transformation is done by a homomorphic function between bicategories which is called *bifunctor*. Let for example

$$HalfAdder = (\vdash \ominus \neg \ominus 1_1^{\Phi}) \Phi (1_1^{\Phi} \ominus \ulcorner \ominus + \ominus \dashv) \Phi (AND \ominus EXOR)$$

$$FullAdder = (HalfAdder \ominus 1_1^{\Phi}) \Phi (1_1^{\Phi} \ominus HalfAdder) \Phi (OR \ominus 1_1^{\Phi})$$

be two bicategorial expressions. *HalfAdder* is described over the set of basic wiring elements plus the basic gates *AND* and *EXOR*. *FullAdder* also uses basic wiring elements, but beside the basic gate *OR* it contains the cell *HalfAdder* which itself is given by a bicategorial expression. In order to get a description of *FullAdder* over the set of basic gates, we have to substitute each occurrence of *HalfAdder* by the corresponding bicategorial expression. This substitution step is also called *expansion* (applying the bifunctor) because we expand the macro *HalfAdder* by its representation. The following figure 4.4 illustrates the expansion step.

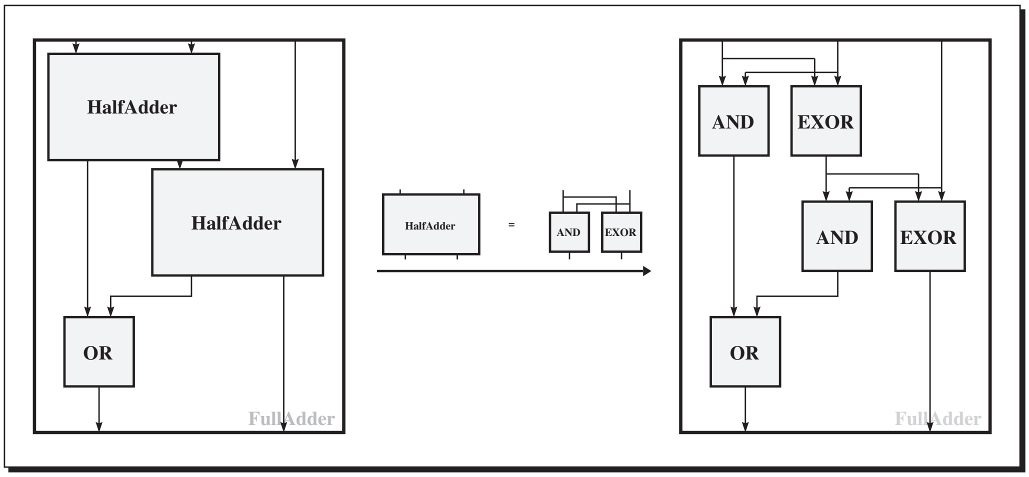


Figure 4.4: Refinement operator for the example of a *FullAdder*

With this method of discrete hierarchy we still have problems in describing very large circuits. Moreover we can only specify fixed realizations of a circuit. In order to get a more flexible specification level we now allow parameterizable net variables and bicategorial expressions. On the one hand this gives us the possibility to describe whole families of circuits with one fixed set of net equations. On the other hand we can setup recursive specifications which lead to very short and handy descriptions for regular circuits, such as adders, multipliers, sorting networks or memory, etc.

The following example demonstrates the use of parameterized net variables:

$$Tree[0] = EXOR$$

$$Tree[i] = (Tree[i-1] \ominus Tree[i-1]) \obar OR$$

This recursive system of net equations has a free parameter i which can be set to an arbitrary nonnegative integer value. If for example we set $i = 3$, then we can apply the expansion functor from above, until we get a bicategorial expression which only contains basic gates as active components. This will take the following steps:

$$\begin{aligned}
Tree[3] &= (Tree[2] \ominus Tree[2]) \obar OR \\
&= ((Tree[1] \ominus Tree[1]) \obar OR \ominus (Tree[1] \ominus Tree[1]) \obar OR) \obar OR \\
&= (((Tree[0] \ominus Tree[0]) \obar OR \ominus (Tree[0] \ominus Tree[0]) \obar OR) \obar OR \ominus \\
&\quad ((Tree[0] \ominus Tree[0]) \obar OR \ominus (Tree[0] \ominus Tree[0]) \obar OR) \obar OR) \obar OR \\
&= (((EXOR \ominus EXOR) \obar OR \ominus (EXOR \ominus EXOR) \obar OR) \obar OR \ominus \\
&\quad ((EXOR \ominus EXOR) \obar OR \ominus (EXOR \ominus EXOR) \obar OR) \obar OR) \obar OR
\end{aligned}$$

The graphical representation of $Tree[3]$ is given in figure 4.5. This simple system of two net equations describes every binary comparison tree for two operands of bit–length 2^i. The result of the circuit is 1, if the two binary numbers differ in at least one bit.

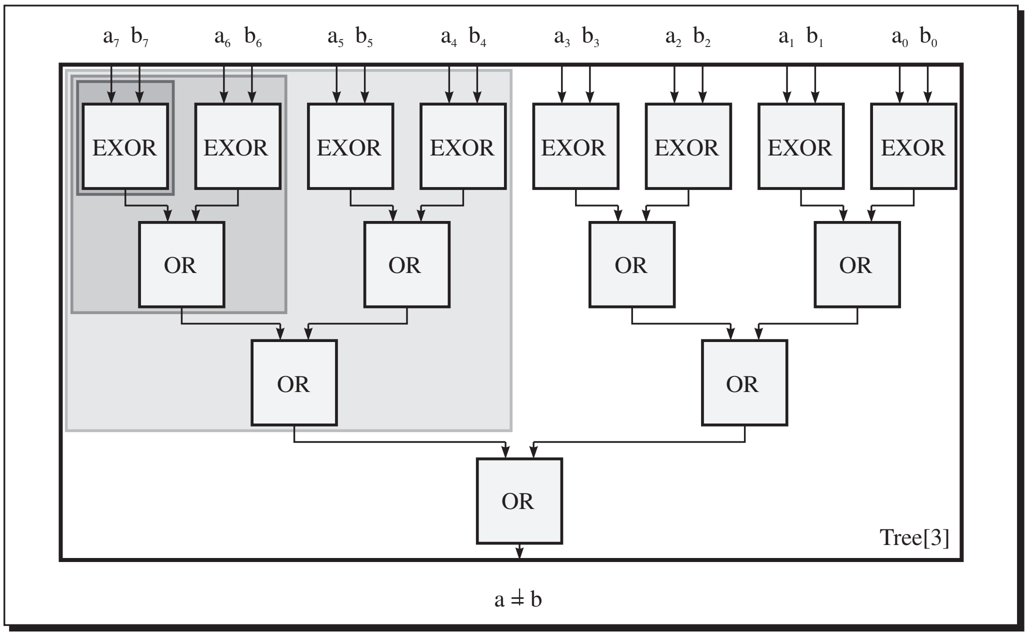


Figure 4.5: Expanded representation of the binary comparison tree *Tree*[3]

This small example shows that the method of recursive net equation systems enables the designer to specify even large circuits by a small number of equations. But the input via bicategorial expressions has some disadvantages:

- the designer creates such an expression after drawing the net on a sheet of paper and then slicing it according to the rules of the calculus.
- for small nets he can already get uncomfortably large expressions.
- short modifications in the circuit may cause tremendous changes in the expressions. Usually they are newly calculated which tends not to be an interactive way of working.
- the initial input may contain a lot of errors, especially if the circuit contains complex wiring components.
- the communication with other design tools is difficult, because the designer has to setup the relation between the calculated results and the given input himself.

To overcome these disadvantages the CADIC system has a graphical input tool, a schematic editor which supports all the features of the underlying mathematical calculus, especially the parameterized and recursive design methodology. In the following sections of this chapter we will introduce the work with this graphical editor of CADIC . We begin our session with some small examples which will show you the basic functions of the editor. During the main part of the chapter we will design an arithmetical circuit as a realistic example to demonstrate, how the user can setup flexible specifications of circuits.

4.3 Getting Started

⤳ 3 If you have followed the instructions given in chapter 3 , then you will see the graphical user interface with its components. In the menuline the topmost menu is displayed which contains a push button for every tool that is integrated in the current graphical environment. From this menu you should select the entry `Schematic Editor` in order to activate the submenu for the graphical editor. After pressing the left mouse button the contents of the menuline changes and the functions of the graphical editor are displayed. These functions are grouped according to the objects they will affect.

In the following design examples we will demonstrate the use of the most important editor functions step by step. A user manual for all editor functions is given in chapter 12. ⤳ 12

4.4 Graphical Specification

From the functional description of a circuit we will now directly derive the schematic inputs for the graphical editor of CADIC . The procedure of entering a schematic usually takes the following basic steps:

- ☐ open a new schematic sheet with an appropriate name and parameter list
- ☐ select and manipulate basic cells or parameterized macros

- ☐ connect the cells, macros and the schematic border with wires
- ☐ enter and place comments for the documentation of the schematic
- ☐ save the schematic

The sequence of these steps can be changed at some positions. You can enter some cells, connect them with wires, add new cells, save the schematic during the session, etc. Especially you can delete cells, wires or comments in the case of mistakes.

In the following sections we will concentrate on these basic steps of drawing a schematic. During this sample session you will especially learn how to open and save schematics, enter, resize, copy, move, rotate and rename basic cells and parameterized macros, enter and delete wires, enter comments and enter additional equations about wire variables. Beside these operations the graphical editor offers more functionality, which will be explained in detail in chapter 12.

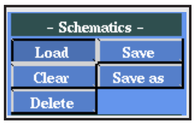
4.5 The First Design: A HalfAdder

In our first design we want to show the graphical input for the halfadder given by the equation

$$HalfAdder = (\vdash \ominus \neg \ominus 1^{\obar}_{1}) \obar (1^{\obar}_{1} \ominus \ulcorner \ominus + \ominus \dashv) \obar (AND \ominus EXOR)$$

Opening a New Schematic

$\rightsquigarrow$ 12.1,12.2.1



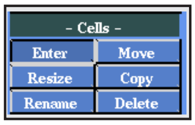
The first step we have to do is to open a new schematic sheet. This is done by selecting the entry `Load` from the submenu `-Schematics-`. You will see a table of the schematics in the directory given by the environment variable `DAGDIR`. At this point of the session this table will only contain a single entry labelled `**** New ****` as it is shown in figure 4.6. Select this entry by moving the pointer onto it and press the left mouse button as soon as it is highlighted. After that a text input window will appear on your screen and you have to specify the name of the new schematic.

```
Please enter New Schematic Name:  <HalfAdder    >
```

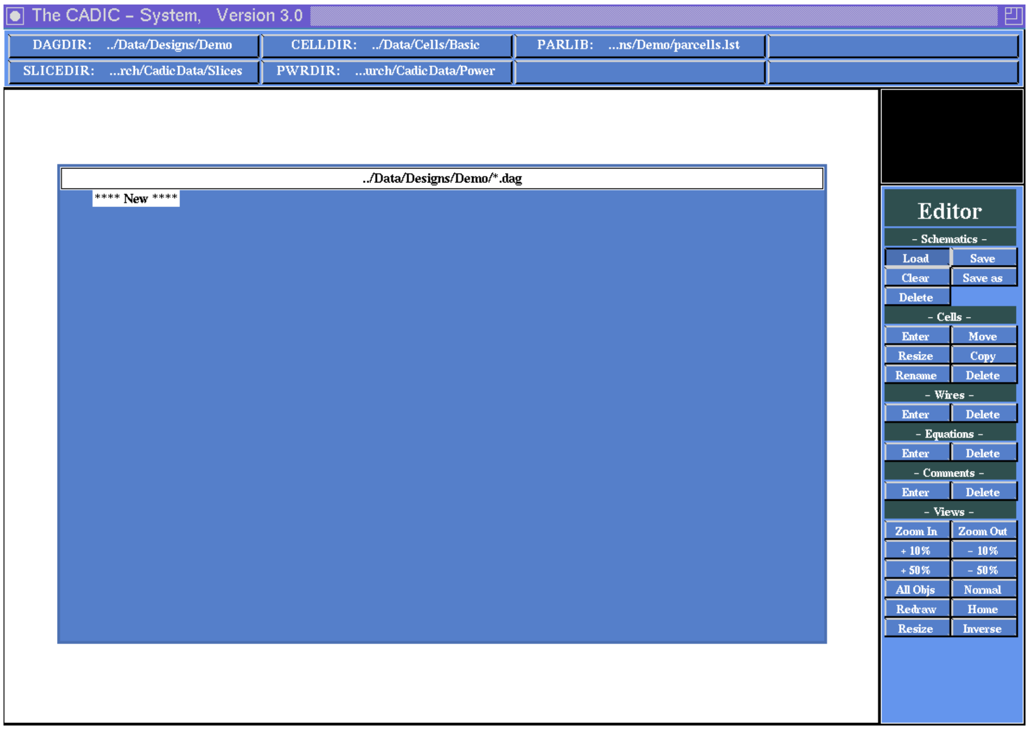
Type in the text `HalfAdder` and press the return key. Then you have opened a new schematic which will first be given by an empty frame. Note, that the name in the upper right field of your environment has changed to `HalfAdder` which is the name of the new schematic.

Entering Basic Cells

⇝ 12.1,12.3.1


- Cells -
Enter
Move
Resize
Copy
Rename
Delete

In the next step we select the appropriate cells for the specification of the halfadder and place them within the given frame. For this purpose select the entry `Enter` from the submenu `-Cells-` in the menuline. After activating this point with the left mouse button, you will see three new windows upon your working area. Each of these windows contains a list of cells of a certain type (c.f. figure 4.7).



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

../Data/Designs/Demo/*.dag

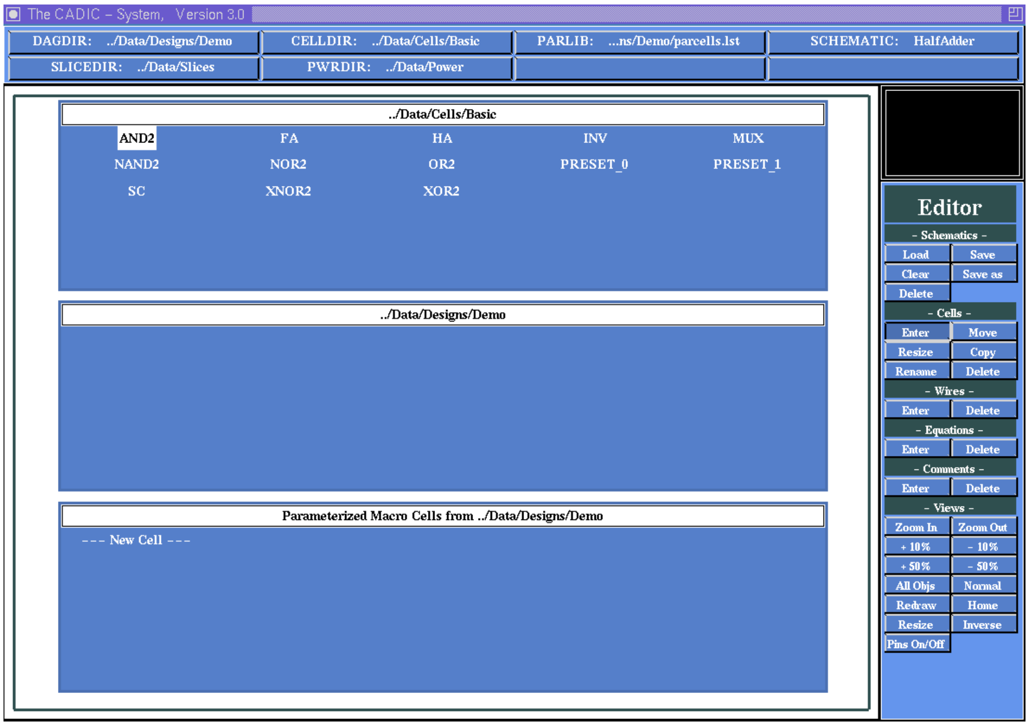
**** New ****

Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.6: Dialog for opening a new schematic

In the upmost window you see the names of the cells in the basic cell library which are located in the directory described by the environment variable

`CELLDIR`. The second window shows the names of macro cells which have been defined by a previous input of a schematic. At this point of the session the macro list is empty. In the third window the names of the parameterized macro cells are listed which have been used during the current session. Actually this list contains one single entry labelled `--- New Cell ---`.



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: HalfAdder
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

../Data/Cells/Basic

AND2	FA	HA	INV	MUX
NAND2	NOR2	OR2	PRESET_0	PRESET_1
SC	XNOR2	XOR2		

../Data/Designs/Demo

Parameterized Macro Cells from ../Data/Designs/Demo

--- New Cell ---

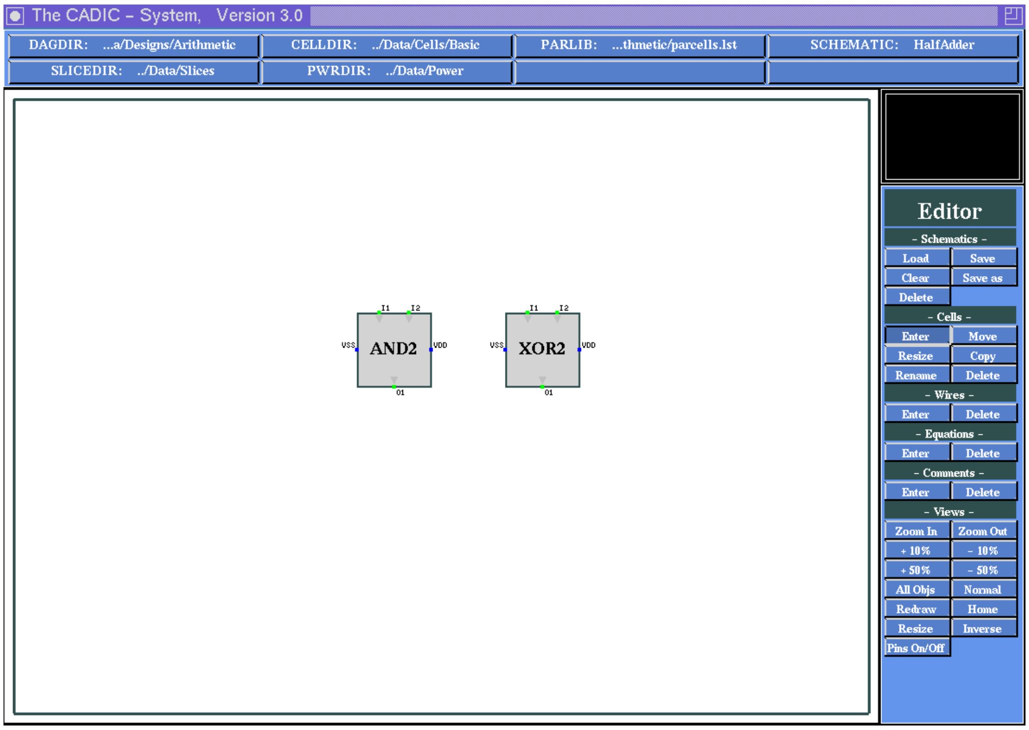
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.7: Dialogs for cell and macro selection

If you look at the equation for the halfadder, the first cell we need is and AND gate which computes the carry of the two inputs. Move the pointer onto the name `AND2` within the first window and press the left mouse button if the name is highlighted. The cell list windows will be closed and you see the shape of an AND gate in the workarea which will follow the pointer motion within the workarea. Now you can pick the right position to place it (c.f. figure 4.8).

If you drop the gate by pressing the left mouse button, it is redrawn with two input pins at its northern border and an output pin at its southern border, indicated by green points. The pins are labelled with the names `I1`, `I2` and `O1`. Small arrows indicate the directions of the pins.

The gate has two more pins called `VDD` and `VSS` (blue points), which represent the power/ground supply ports. These pins are uninteresting for the logical function of our circuit and we will not use them during the editor session. The power supply nets which will connect all these supply pins are automatically generated by an appropriate tool (c.f. chapter 8). ↝ 8


The CADIC – System, Version 3.0

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: HalfAdder
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

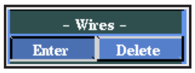
I1
I2
VSS
AND2
VDD
O1
I1
I2
VSS
XOR2
VDD
O1
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.8: Placement of the basic cells for `HalfAdder`

After the placement the cell list windows are popped up again in order to select another gate. Now move the pointer to the entry `XOR2` to get an EXOR gate for two inputs. We need this gate to compute the sum modulo 2 of the two input bits, so place it right beside the AND gate as it is shown in figure 4.8. Because this is the last gate we need for the halfadder, you should abort the cell selection mode by pressing the right mouse button within one of the three list windows.

Entering Wires

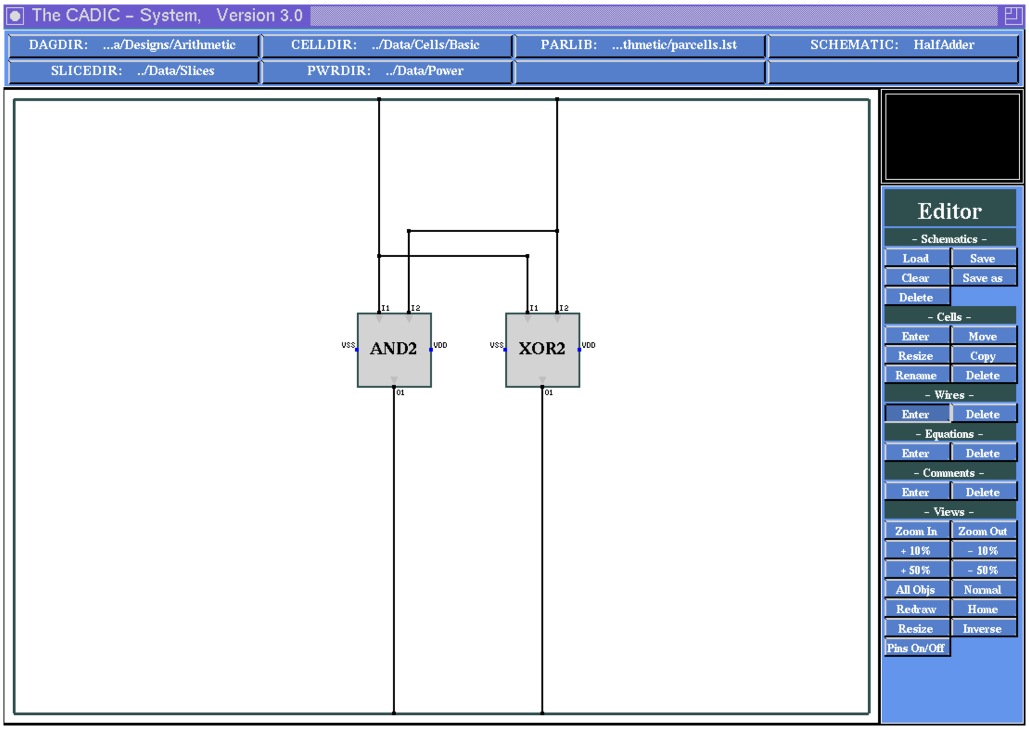
⤳ 12.4.1



To draw the wiring of `HalfAdder` select the entry `Enter` from the `-Wires-` submenu. First we connect the output pins of the two gates directly with the southern border of the schematic. Move the pointer near to the output pin `O1` of the `AND2` gate. If the distance to the pin is less than a certain tolerance value the crosshair cursor will snap to the pin and you can fix the start point of the wire by pressing the left mouse button. Now move vertically down to the southern border of the schematic. During the pointer motion you will notice a grey rubberbanding line following the pointer. This line indicates the current connection which will be established, if you press the left mouse button to fix the end point of the wire. In some cases the rubberbanding line will dissappear in order to show you that the current connection is illegal. For example this is the case, if the connection would cut a cell or the schematic border. Pressing the left mouse button with an invisible rubberband would result in nothing but the selection of a new start point.

If you move the pointer near to the southern border of the schematic it will snap again to the border line and you can fix the end point by pressing the left mouse button. The newly created wire is established and redrawn as a yellow connection between the pin and the schematic border. In this case of connecting a pin of a basic cell the width of the wire is automatically set to 1. Later you will see that the width of a wire can also be set to flexible values.

After connecting the output pin `O1` of the `XOR2` gate with the southern border of the schematic we turn to the input pins. Connect the left input `I1` of the `AND2` gate as well as the right input `I2` of the `XOR2` gate directly with the northern border of the schematic. These two wires represent the inputs a and b of the halfadder. As shown above we have to feed these values into both the `AND2` and `XOR2` gate, i.e. we branch the wires and connect them with the appropriate input pins. To do this move the pointer onto the left wire. You can notice that it snaps to the wire if the distance is less than a certain tolerance value. Press the left mouse button to fix



The CADIC – System, Version 3.0

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: HalfAdder
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

AND2
XOR2
I1
I2
VSS
VDD
O1
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

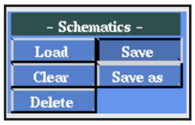
Figure 4.9: Wiring for `HalfAdder`

the start point and move the pointer to the left input pin of the `XOR2` gate. The connection is indicated by a rubberbanding line between the start point and the current position of the pointer. If you select an illegal position the rubberbanding line will disappear. If you have setup the right connection to the left input of the `XOR2` gate press the left mouse button to fix the end point of the wire. The yellow dot at the start point of the new wire indicates a branching of the previous vertical wire. Implicitly the branch node is considered to be a basic operation in the underlying calculus. In this special case we have entered an east–branch.

In the same way you can connect the right vertical wire with the right input pin of the `AND2` gate. You should have entered a schematic similar to that shown in figure 4.9. Terminate the enter wire mode by pressing the right mouse button two times (the first time you press the right mouse terminates the selection of an end point and turns to the selection of a new start point).

Saving the Schematic

⇝ 12.2.2



Save the schematic by selecting the entry `Save` from the `-Schematics-` submenu. The graphical input is then written to the file `HalfAdder.dag` in the directory given by the environment variable `DAGDIR`.

4.6 Going into Hierarchy: A FullAdder

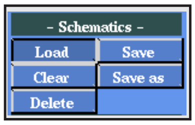
In this section we will specify a small circuit which consists of two hierarchy levels. According to the description from above

$$FullAdder = (HalfAdder \ominus 1_1^{\Phi}) \mathbin{\Phi} (1_1^{\Phi} \ominus HalfAdder) \mathbin{\Phi} (OR \ominus 1_1^{\Phi})$$

will we build a fulladder with the help of the halfadder from the previous section.

Opening a New Schematic

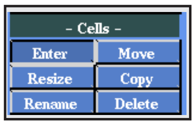
⇝ 12.2.1



We call this new schematic `FullAdder` and open it by following the steps in section 4.5. Note that the list of schematics now contains the entry `HalfAdder` beside the position of `**** New ****`. After selecting a new schematic you should type in the name `FullAdder` and press return. Now we have created an empty frame for entering the fulladder design.

Entering Macro Cells

⇝ 12.1,12.3.1

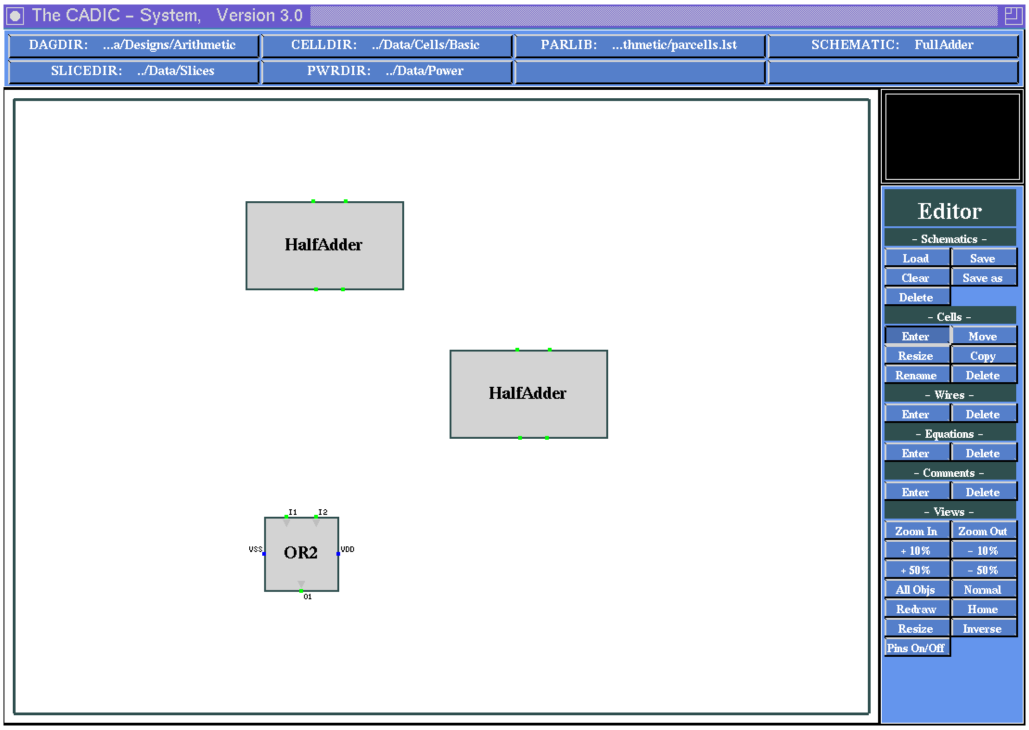


In the next step we select the appropriate cells for the specification of the fulladder and place them within the given frame. For this purpose select the entry `Enter` from the submenu `-Cells-` in the menuline. As shown above we need two instances of a halfadder together with an instance of an OR gate. The halfadder is described by its own schematic. Therefore you will now see an entry `HalfAdder` in the second cell selection window. This is the window for discrete macro cells, i.e. macro cells which do not depend on any free parameter.

Move the pointer onto this entry and select it by pressing the left mouse button after it is highlighted. Then the cell selection windows are popped down and you notice a cell of a certain size following the pointer motion within the workarea. Place it as it is shown in figure 4.11 and fix the position by pressing the left mouse button. After that the cell selection windows are popped up again in order to choose another cell.

Select a second instance of the halfadder and place it lower right to the first one (c.f. figure 4.10). Finally select an OR gate (`OR2`) from the first cell selection window which contains the list of basic cells. Place it at the position shown in figure 4.10 below the second half adder.

Now terminate the cell selection mode by pressing the right mouse button within one of the selection windows.


The CADIC – System, Version 3.0

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: FullAdder
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

HalfAdder
HalfAdder
OR2
I1
I2
VSS
VDD
O1
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.10: Using the macro `HalfAdder` to construct a fulladder

Before we continue you should take a short look at the macro cells and compare them to the basic cells. You notice that the macro `HalfAdder` has two pin connectors at its northern border and southern border. These pins correspond to the wires we have connected with the schematic borders of the corresponding schematic for the halfadder. The position of the pins is relative to the position of the wire connection at the schematic frame.

In contrary to the basic cells there are no pin names shown for the macro cells. This does not mean that the pin connectors have no names, but they are automatically generated and omitted for reasons of simplicity. The pin names of macro cells contain the exact position of the corresponding wire. For example the upper left pin of `HalfAdder` has the name `n[0,0]` which means that it belongs to the northern border and represents the interval of wires from `0` to `0` inclusively, i.e. it represents the first wire connected to the northern border and this wire has the width 1. For the conventions of naming pins of macro cells see also 12.3.1.

↝ 12.3.1

Entering Wires

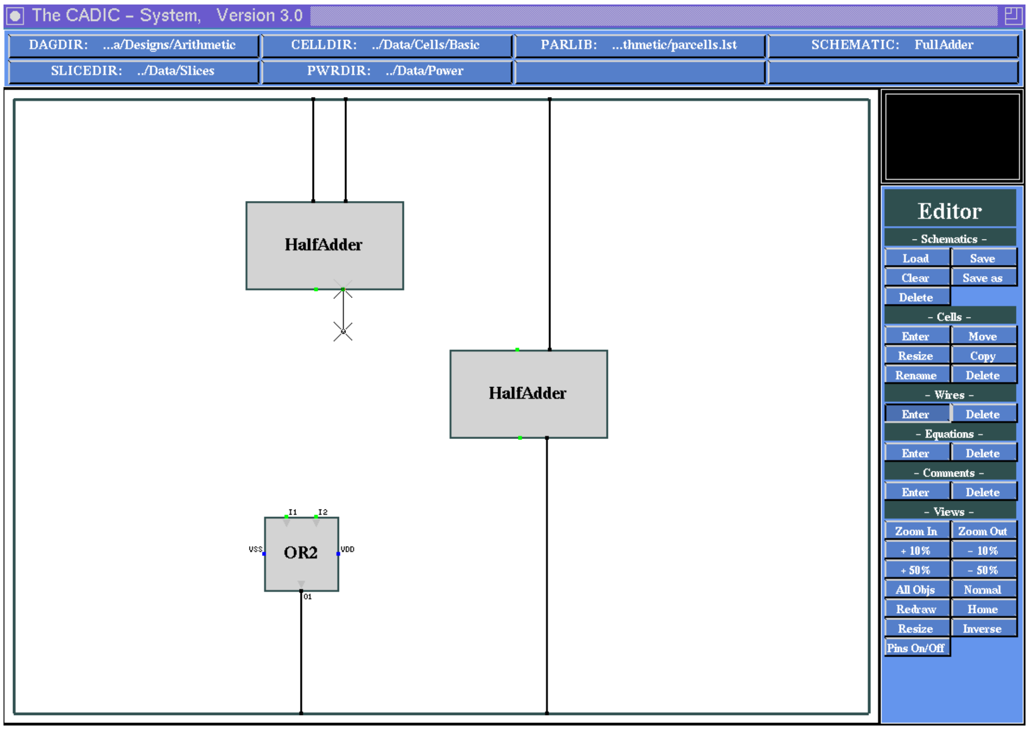
↝ 12.4.1

- Wires -
Enter
Delete

To draw the wiring of `FullAdder` select the entry `Enter` from the `-Wires-` submenu. First we connect the output pin of the OR gate and the right output pin (`s[1,1]`) of the lower halfadder macro directly with the southern border of the schematic. Move the pointer near to the output pin of the `OR2` gate. If the distance to the pin is less than a certain tolerance value the crosshair will snap to the pin and you can fix the start point of the wire by pressing the left mouse button. Move vertically to the southern border of the schematic, dragging the grey rubberbanding line, and fix the end point in the same way. As mentioned above the width of this wire is automatically set to 1.

Now move the pointer to the right pin at the southern border of the lower halfadder and fix the start point with the left mouse button. Draw the rubberbanding line vertically down to the southern schematic border and fix the end point. Because the pin of the macro cell represents one single binary value, the width of this wire is also automatically set to 1.

After connecting these output pins with the southern border of the schematic we turn to the input pins of the fulladder. Connect the two input pins of the upper left halfadder as well as the right input pin of the lower right halfadder directly with the northern schematic border. These three wires represent the inputs of the fulladder.



The CADIC – System, Version 3.0

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: FullAdder
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

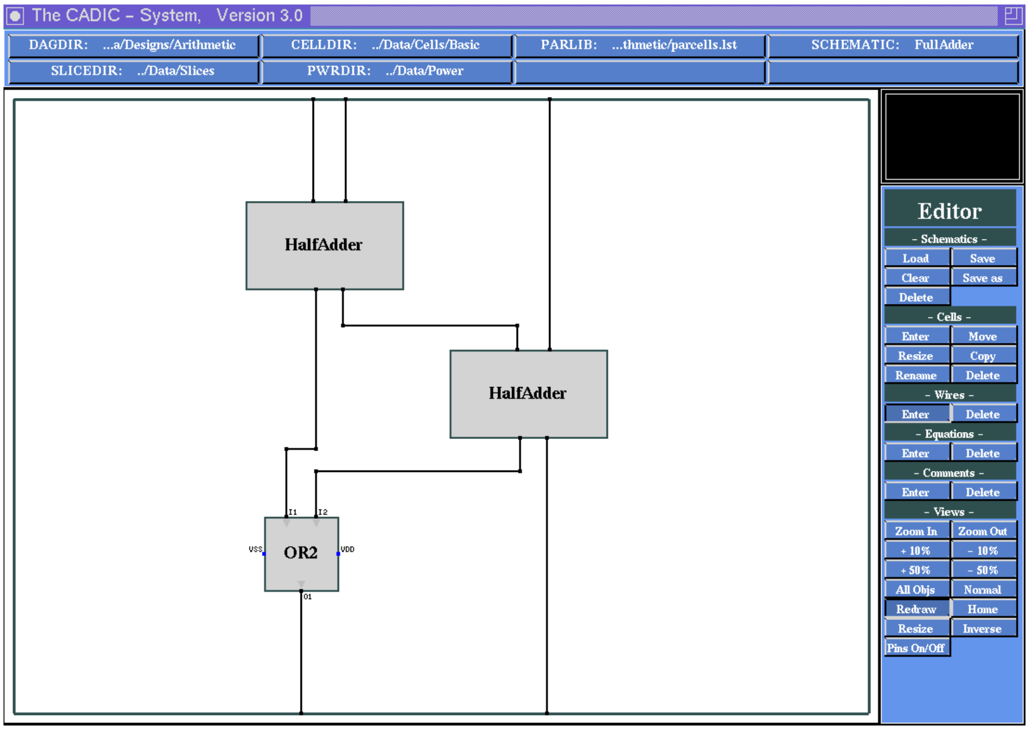
HalfAdder
HalfAdder
I1
I2
VSS
OR2
VDD
O1
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.11: Wiring of `FullAdder` and creating a knee for a connection between two pins

According to the sketch from figure 4.4 we still have to connect the right output pin of the left halfadder with the left input pin of the right halfadder and the left output pins of both halfadders with the input pins of the OR gate.

To do this wiring move the pointer onto the right output pin of the left halfadder. You can notice that it snaps to the pin if the distance is less than a certain tolerance value. Press the left mouse button to fix the start point and move the pointer vertically downwards. The actual connection is indicated by the rubberbanding line between the start point and the current position of the pointer. If you select an illegal position the rubberbanding line will disappear. This is the case if the chosen connection intersects a cell or moves along an already existing wire or the border of the schematic.

In our case we first create a small vertical wire, from which we will complete the connection between the two pins. Fix the end point as it is shown in figure 4.12. This end point now serves as the start point for the next wire, i.e. just move the pointer onto the left input pin of the right half adder. You will notice that the rubberbanding line will automatically create a knee for the proper connection. Fix the end point at the left input pin of the halfadder by pressing the left mouse button.


The CADIC – System, Version 3.0
HalfAdder
HalfAdder
OR2
I1
I2
VSS
VDD
O1
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

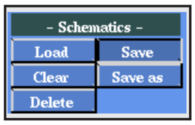
DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: FullAdder
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

Figure 4.12: Final wiring for `FullAdder`

We also apply this method of connecting two pins for the remaining two wires between the left output pins of the halfadders and the input pins of the OR gate. Finally you can terminate the enter wire mode by pressing the right mouse button two times within the workarea.

Saving the Schematic

⤳ 12.2.2


- Schematics -
Load
Save
Clear
Save as
Delete

Save the schematic by selecting the entry `Save` from the `-Schematics-` submenu. The graphical input is then written to the file `FullAdder.dag` in the directory given by the environment variable `DAGDIR`.

4.7 Parameterization: An n Bit Comparison Tree

In this section we will show how you can setup a parameterized specification of a circuit. In contrast to the previous two sections, where we designed a discrete hierarchical circuit, we will now specify a whole family of circuits. The advantage of CADIC is that we can do this with the help of a finite number of graphical inputs.

The parameterization is done with the help of recursive equations, i.e. we use a regularity in the circuit structure to get a compact description (a small number of inputs). Normally a recursive specification consists of basic equations which serve as end conditions of a recursion. These are equations, where at least one parameter of the subcircuit is given by a fixed nonnegative integer value. You can have more than one basic equation for the same subcircuit, for example if there are different basic elements for different initial parameter values. The recursion itself is given by so called general equations. These are equations, where all parameters are free. For each subcircuit there may exist only one general equations. If this is not the case the refinement operation would ambiguous.

The example of the n bit comparison tree, we will show in this section, can be described by two parameterized equations:

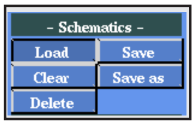
$$Tree[0] = EXOR$$

$$Tree[i] = (\,Tree[i-1] \ominus Tree[i-1]) \Phi \, OR$$

The first equation is a basic equation which serves as the end of the recursion. It handles the case that we have to compare two single bits a_0 and b_0 ($n = 2^0$). The second equation is the general recursive equation, where we reduce the comparison of two $n = 2^i$ bit numbers to the comparison of the upper and lower halfs and combining these intermediate results by an OR gate. In the following sections we will now show, how you can enter a schematic input for these two equations.

4.7.1 Basic Equation for the n Bit Comparison Tree

Opening a New Schematic

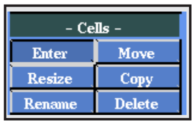


⇝ 12.2.1 We begin with the basic equation for the recursive description of the n bit comparison tree. This will be a compare subcircuit for $n = 2^0$ bit operands. Therefore we call this schematic `Tree[0]` which will be opened following the steps in section 4.5. Note that the list of schematics now contains the entries `HalfAdder` and `FullAdder` from our previous work beside the position of `**** New ****`. After selecting a new schematic you should type in the name `Tree[0]` and press return.

```
Please enter New Schematic Name:  <Tree[0]    >
```

The name of the new schematic is `Tree`. The square brackets introduce a list of parameters which can be used during the specification of the schematic. In this case we have one single parameter which is set to the constant value `0`. In general you can use up to 32 parameters for one schematic. ⇝ 12.1

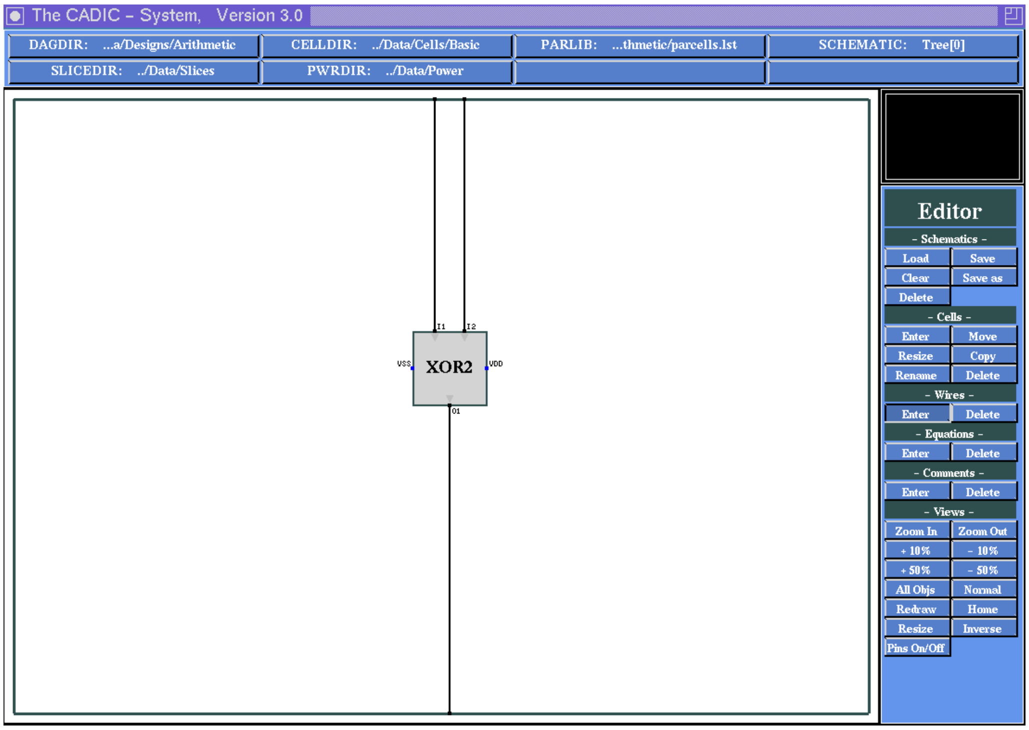
Entering Basic Cells



⇝ 12.1,12.3.1 In the next step we select the appropriate cells for the basic compare element and place them within the given frame. For this purpose select the entry `Enter` from the submenu `-Cells-` in the menuline. After picking this point with the left mouse button, you will see the three cell selection windows upon your working area (c.f. figure 4.7).

The task to compare two single bits a_0 and b_0 is trivial and can be solved with the help of an EXOR gate. The result of this gate is 1, if the input bits are different, and 0, if they are equal.

Select the entry `XOR2` from the first cell selection window and press the left mouse button, if it is highlighted. Move the gate into the middle of the working area and drop it there by pressing the left mouse button again. Because this is the only cell we need in this schematic you can terminate the cell selection by pressing the right mouse button within one of three selection windows.



The CADIC – System, Version 3.0

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: Tree[0]
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

I1
I2
VSS
XOR2
VDD
O1
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.13: The basic equation for the n bit comparison tree

Entering Wires

⤳ 12.4.1

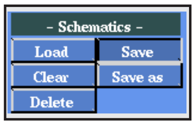
– Wires –
Enter
Delete

To draw the wiring of `Tree[0]` select the entry `Enter` from the `-Wires-` submenu. The wiring in this case is very simple. We just have to connect the pins of the EXOR gate with the corresponding schematic borders. Move the pointer near to the output pin of the `XOR2` gate. If the distance to the pin is less than a certain tolerance value the crosshair will snap to the pin and you can fix the start point of the wire by pressing the left mouse button. Move vertically to the southern border of the schematic and fix the end point in the same way. In the same way connect the input pins of the gate with the northern border of the schematic. The width of all three wires is automatically set to 1.

The schematic should look like that shown in figure 4.13. Finally terminate the enter wire mode by pressing the right mouse button twice within the working area.

Saving the Schematic

↝ 12.2.2

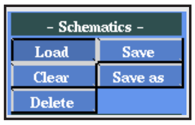


Save the schematic by selecting the entry `Save` from the `-Schematics-` submenu. The graphical input is then written to the file `Tree[0].dag` in the directory given by the environment variable `DAGDIR`.

4.7.2 General Equation for the n Bit Comparison Tree

Opening a New Schematic

↝ 12.1,12.2.1



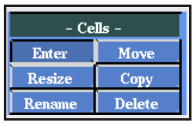
We will now enter the recursive equation of the n bit comparison tree, for which we first have to open a new schematic sheet. After the selection of the entry `Load` from the submenu `-Schematics-` you should type in `Tree[n]`.

As in the case of the basic equation the square brackets introduce the parameter list of the schematic. Here we have one free parameter `n` which can be used within the elements of the schematic as macros cells and wires.

You may use any legal identifier as the name for a parameter. The syntactical structure of an identifier is defined as you may know it from programming languages. ↝ 12.1

Entering and Manipulating Cells

↝ 12.1,12.3.1



In our recursive description we reduce the comparison of two 2^n bit numbers to the parallel comparison of the upper 2^{n-1} and lower 2^{n-1} bits of the numbers. This means that we need two instances `Tree[n-1]` of 2^{n-1} comparison trees within `Tree[n]`.

In this next step we select the appropriate cells for the recursive specification of the comparison tree and place them within the given frame. For this purpose select the entry `Enter` from the submenu `-Cells-` in the menuline.

The input of the parameterized macros for the 2^{n-1} bit comparison trees is done within the third of the cell selection windows. At this point of the session this windows contains a single entry `--- New Cell ---` for the definition of a new parameterized macro cell. Select this in the same way as you have selected to open a new schematic in order to define a new parameterized macro cell. After hitting the left mouse button you will be prompted to enter a new macro name.

```
Please enter Parameterized Name:  <Tree[n-1]    >
```

Type in `Tree[n-1]` to specify a macro for a comparison tree for 2^{n-1} bit numbers.

Note that you can only use the parameters of the schematic to build arithmetic expressions for the parameters of the macro cells. Try to specify a cellname `Tree[k-1]` and the system will respond with the message

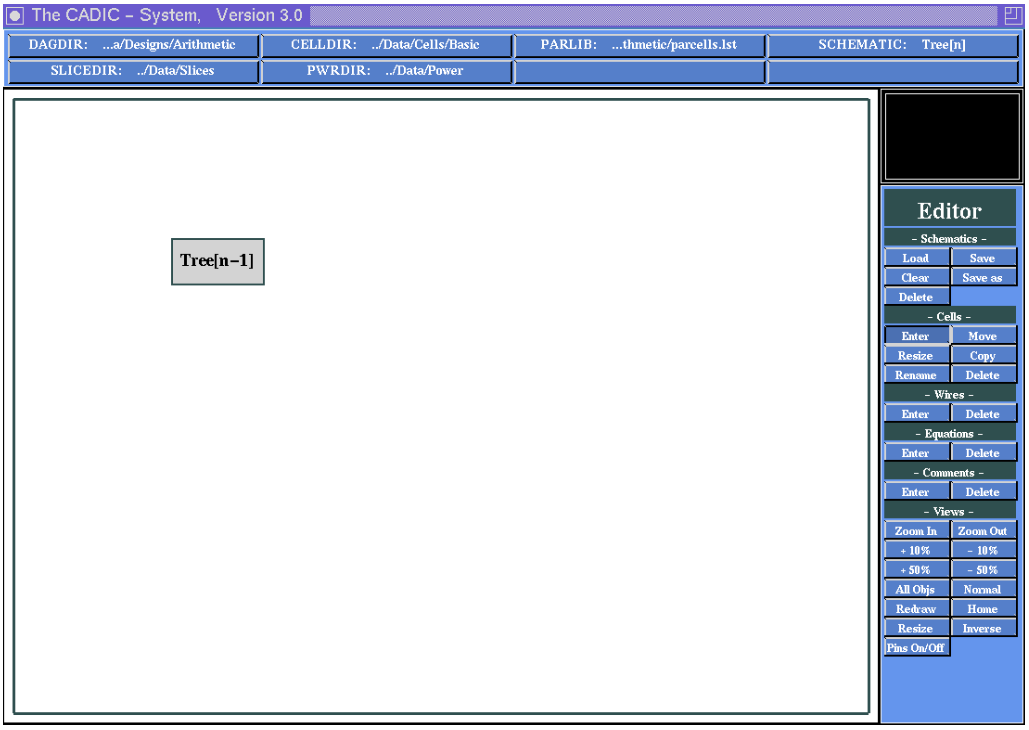
```
        !  Error !  "Tree[k-1]" uses illegal parameter "k"
```

The syntax diagrams for the expressions allowed as parameters of these macro cells are given in chapter 12.

After the specification of a correct cellname you will see the graphical representation of the macro, i.e. a red bounded box with the cellname in its center. You can move this symbol inside the frame and place it at the desired position by clicking the left mouse button. The macro can only be placed on legal positions, i.e. there must not be an overlapping with other macros, wires or the border of the schematic. Try to move the macro onto the schematic frame and you will notice a change of its colour from red to grey. This change indicates the selection of an illegal position. Now place the macro onto the position you see in figure 4.21.

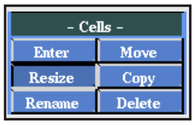
The size of the macro is chosen by default, because the selected parameterized macro represents a whole class of macros. Its exact size can be determined, if the values for the parameters are known. For the abstract specification level of the editor the size of a cell or a macro is not important. In order to setup a comfortable schematic which is easy to understand, we will now resize the macro, such that it gives us an impression of the relative sizes. Before we can perform the resize operation, we have to terminate the enter cell function (third mouse button in one of the cell selection windows).

⇝ 12.3.3 For the resize operation push the button `Resize` in the `-Cells-` submenu.


The CADIC – System, Version 3.0
Tree[n–1]
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: Tree[n]
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

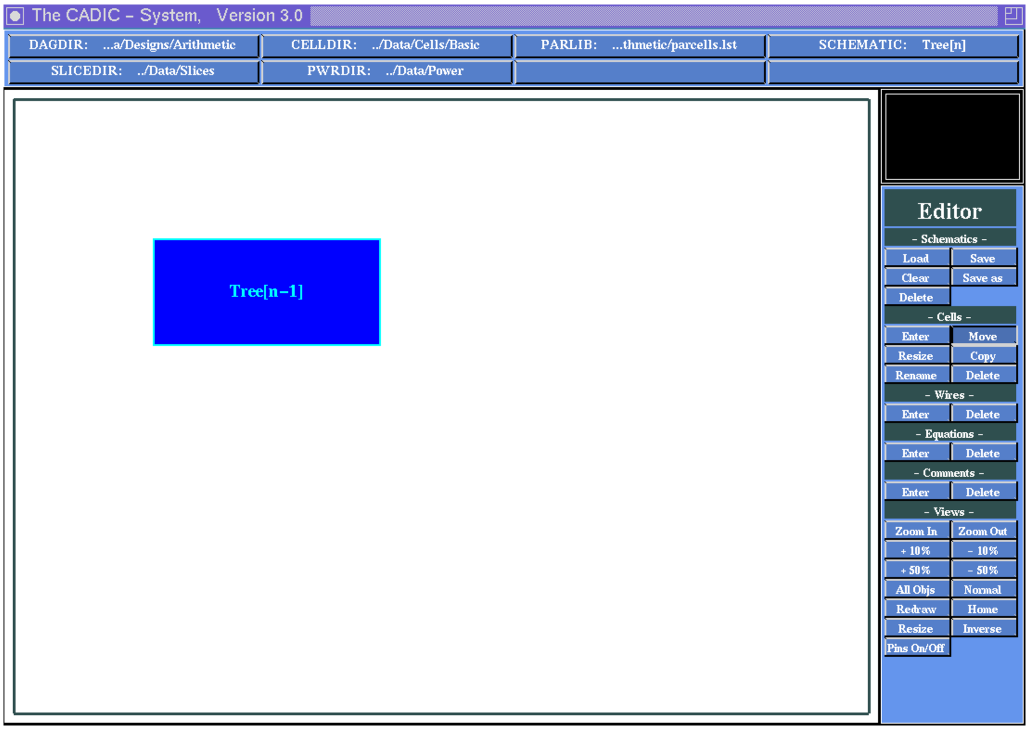
Figure 4.14: Placing a parameterized macro cell for the recursive specification of the comparison tree


– Cells –
Enter
Move
Resize
Copy
Rename
Delete

Then you are in the cell selection mode and can pick the desired cell within the workarea. Move the pointer onto the workarea and you will see the macro being highlighted (its colour changes from grey to blue). In cell selection mode always the macro nearest to the pointer will be highlightet, you need not be inside it. At this point of our session cell selection is trivial because we only have a single macro within the workarea. The selection is done by clicking the left mouse button.

After the selection you can draw a rubberbanding frame which is anchored in the upper left corner of the macro and represents its new size. During the resize operation you need not hold down the left button. Just move the pointer to the desired position and confirm the new size by clicking the left mouse button or cancel the operation with the right mouse button (you will be returned to cell selection mode).

If you confirm a new size for the macro you will be in cell placement mode.



The CADIC – System, Version 3.0

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: Tree[n]
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

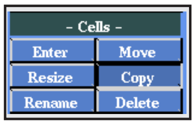
Tree[n−1]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.15: Resizing a parameterized macro cell

The reason for this are possible overlappings of the resized macro with its neighbours or with wires. You have to choose a new position for it which is only possible within the given constraints (i.e. no overlapping with macros, wires or the border). Sometimes it can happen that you have no enough free space in your workarea to perform the resize operation without creating overlaps with other cells. In this case you should terminate the resize operation and enlarge the workarea with the help of the functions from the submenu `-Views-` which are explained in section 12.7.1. ⇝ 12.7.1

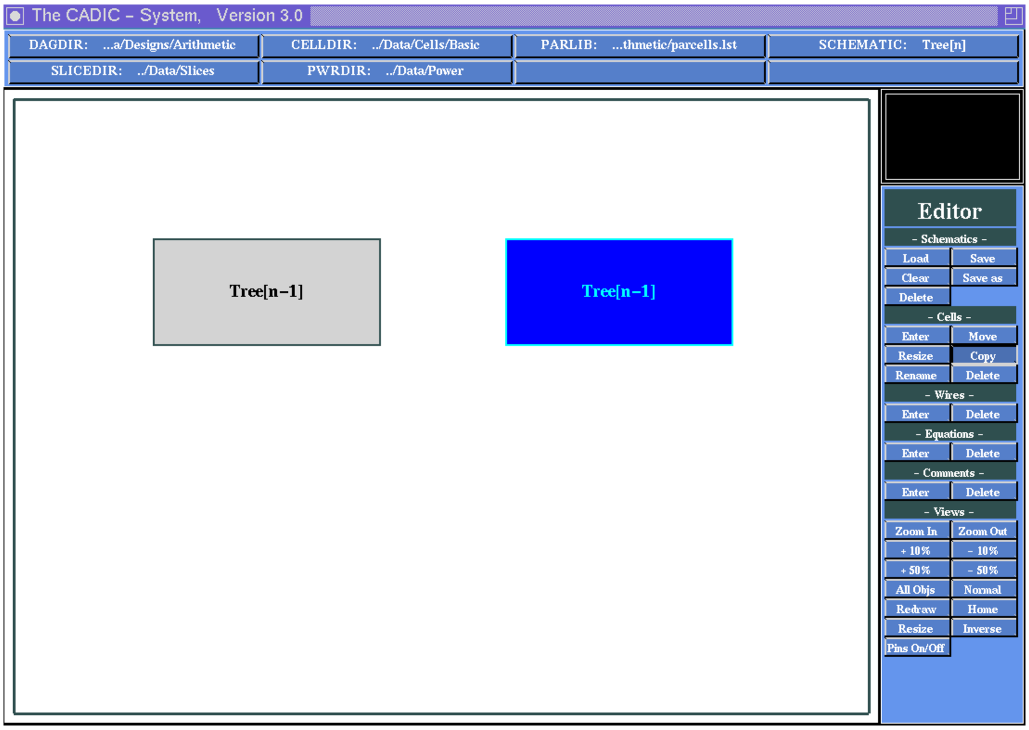
Move the resized macro to the previous position and confirm it with the left mouse button. Now your schematic should look like that shown in figure 4.15.

⇝ 12.3.4 As described above the 2^n bit comparison tree simultaneously compares the upper 2^{n-1} and the lower 2^{n-1} bits. We now have positioned the macro `Tree[n-1]` for the upper half of the operands. A second instance of this macro will be needed for the lower half. We could create this instance by



- Cells -
Enter
Move
Resize
Copy
Rename
Delete

following the above steps (selecting a parameterized macro, placing and resizing it). But the editor offers a much simplier way to perform this operation. Push the entry `Copy` from the `-Cells-` submenu. This enables the cell selection mode where you can select an instance to be doubled. Press the left mouse button and you will get a second macro labelled `Tree[n-1]` which has the same size as our first macro. Place it within the workarea in order to have the schematic look like that in figure 4.16. During the placement operation you can watch the macro changing its colour, if it is placed on the first `Tree[n-1]` or on the border. In this case you cannot drop it, i.e. pressing the left mouse button has no effect. You can cancel the operation by pressing the right mouse button. The copy will dissappear and you will return to cell selection mode. Pressing the right mouse button again will terminate the whole copy operation.



The CADIC – System, Version 3.0

DAGDIR: ...a/Designs/Arithmetic

CELLDIR: ../Data/Cells/Basic

PARLIB: ...thmetic/parcells.lst

SCHEMATIC: Tree[n]

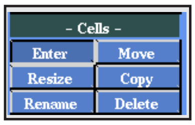
SLICEDIR: ../Data/Slices

PWRDIR: ../Data/Power

Tree[n–1]
Tree[n–1]
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.16: Creating a copy of an exisiting macro cell

⤳ 12.1,12.3.1



– Cells –
Enter
Move
Resize
Copy
Rename
Delete

Finally we need an OR gate to combine the results of the two 2^{n-1} comparison trees. For this purpose select the entry `Enter` from the submenu `-Cells-` in the menuline again. Select an OR gate (`OR2`) from the first cell selection window which contains the list of basic cells. Place it at the position shown in figure 4.17 below the two 2^{n-1} bit comparison trees, such that we get a tree–like structure. Now terminate the enter cell mode by pressing the right mouse button within one of the three selection windows.

Entering Wires

↝ 12.4.1

- Wires -
Enter
Delete

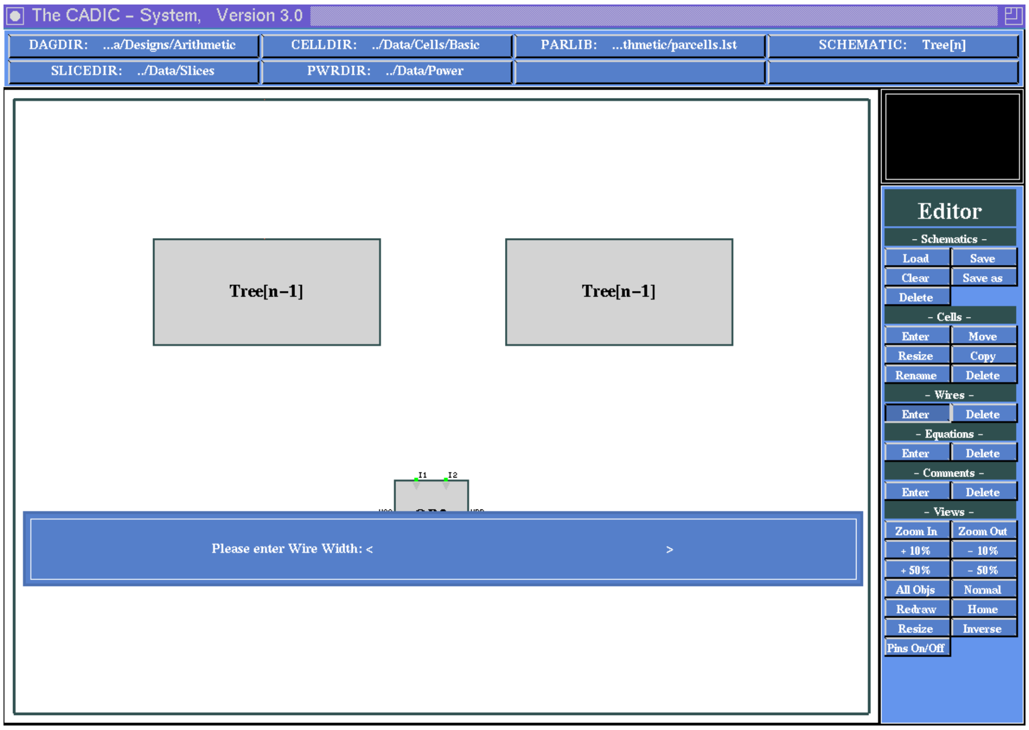
In this section we show how to connect the modules with wires. The main point you will observe during this procedure is that because of the parameterized description the lines drawn will have variable widths. The width of a line specifies the number of parallel wires contained in this bundle.

We start with the left `Tree[n-1]` which compares the upper half of the bits of the the operands. First select the point `Enter` from the `-Wires-` submenu and move the pointer into the workarea. The cursor has changed to a crosshair and you can select the start point for a new wire. Move the pointer to the northern side of the left `Tree[n-1]` macro and position it to the middle of this side. If you move the pointer slowly toward the border of the macro you can observe that it snaps onto the border, if the distance becomes less than a certain tolerance value. This indicates that you have selected a special position within the schematic. The same behaviour can be noticed if you reach the frame, a pin of a cell or another wire. If the pointer has snapped to the border of the macro press the left mouse button to fix the start point of the new wire. Now move the pointer vertically towards the northern border of the schematic. You will see that it is followed by a grey rubberband line which indicates the position of the new wire. The cursor snaps to the border if the distance is small enough.

Now you can fix the end point of the wire by pressing the left mouse button again. The grey rubberband line will disappear and you will be prompted to enter the width of this wire.

```
Please enter Wire Width:  <2^n      >
```

As mentioned above the width of a wire represents the number of parallel lines within this bundle. In our case this wire describes the input line to


The CADIC - System, Version 3.0
Tree[n-1]
Tree[n-1]
I1
I2
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: Tree[n]
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

Please enter Wire Width: < >

Figure 4.17: Entering a wire of parameterized width

the first comparison tree for the upper half of the operands. The width of a single operand is given by the parameter 2^n. The left `Tree[n]` gets two operands of length 2^{n-1}, i.e. the number of single wires is 2^n which you can type in in the form `2^n`. You will see the new wire as a yellow line from the border of the `Tree[n-1]` to the border of the schematic. In the middle of the wire you can notice the description of its width annotated by a small diagonal line with the appropriate expression (c.f. figure 4.18).

Note that you can only use the parameters of the schematic to build arithmetic expressions or wire variables. The exact syntax for these expressions is given by the diagrams in appendix. ⇝ 12.1

We proceed with the other modules. The right instance of `Tree[n-1]` will also be connected with the northern border of the schematic. You can draw this wire in the same way as you did for the left `Tree[n-1]`. Type `2n̂` for its width in order to denote that it describes the wires for the lower parts of the operands.

Now we turn to the southern borders of the two `Tree[n-1]`. The output of both has to be connected with the input pins of the OR gate. Move the pointer near to the middle of the southern border of the left `Tree[n-1]` and fix the start point of the new wire. Because the connection to the input pin of the OR gate has to be drawn with two knees, we first create a small vertical wire segment. After fixing the end point of this segment you are prompted to enter the width of this wire, because the system can not automatically determine the width of this wire (it is connected to a parameterized macro cell).

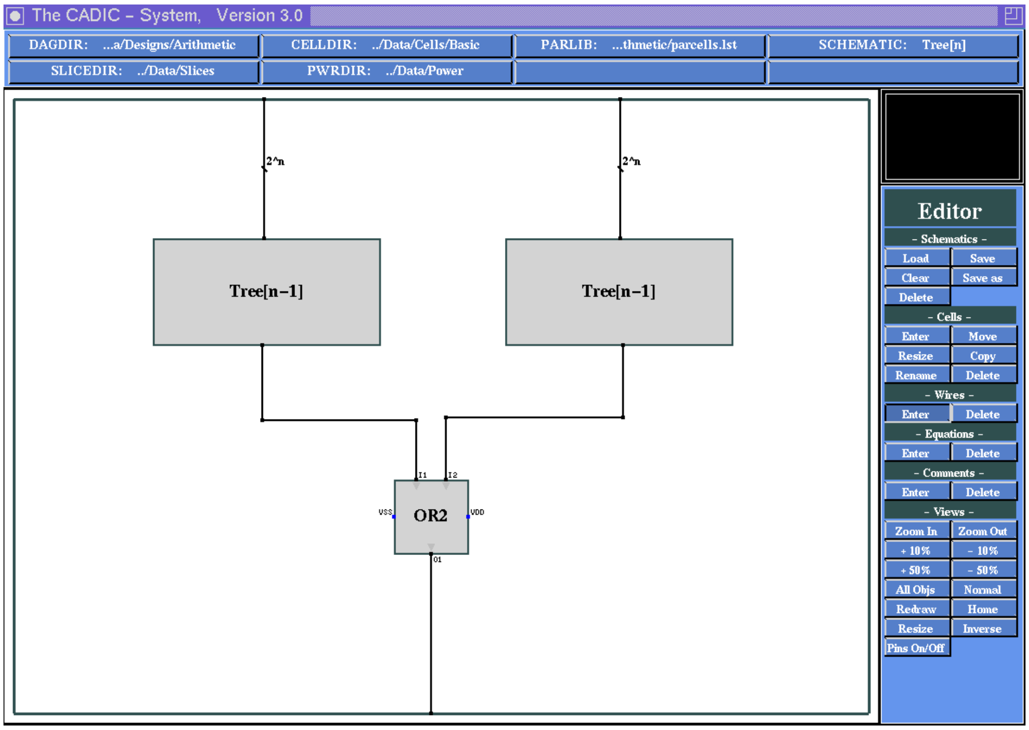
```
Please enter Wire Width:  <1       >
```

Enter `1` because the result of the 2^{n-1} bit comparison tree is a single binary value. Now continue the drawing of this wire by moving the pointer to the left input pin of the OR gate. The rubberbanding line will automatically create the second knee for the correct connection. If you fix the end point you are not prompted for the width of this wire, because you continued an already existing wire.

The entering of the wire width `1` from above can be avoided, if we start the wire at the input pin of the OR gate as we will do this for the second `Tree[n-1]`. Fix the start point at the right input pin of `OR2` and move the pointer a short distance upwards. If you fix an end point here the wire automatically gets the width 1. Now continue the wire to the middle of the southern border of the right `Tree[n-1]`. Press the left mouse button again and fix the end point of the connection.

Finally connect the output pin of `OR2` with the southern border of the schematic as the output of the whole comparison tree `Tree[n]`.

To leave the enter wire mode press the right mouse button twice. The first button press will deactivate the selection of a wire end point and return to the selection of new start point. The second button press will abort the enter wire mode and return to the menu selection mode.

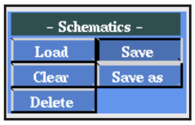

The CADIC – System, Version 3.0
2^n
2^n
Tree[n–1]
Tree[n–1]
I1
I2
VSS
OR2
VDD
O1
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

DAGDIR: ...a/Designs/Arithmetic
CELLDIR: ../Data/Cells/Basic
PARLIB: ...thmetic/parcells.lst
SCHEMATIC: Tree[n]
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

Figure 4.18: Final wiring for the 2^n bit comparison tree

Saving a Schematic

⤳ 12.2.2


– Schematics –
Load
Save
Clear
Save as
Delete

Now that we have drawn all elements of the general recursive equation for the 2^n bit comparison tree we must save the schematic. This is simply done by selecting the entry `Save` from the `-Schematics-` submenu. The schematic will be saved to the file `Tree[n].dag` in the directory given by the environment variable `DAGDIR`.

4.8 Design of an n Bit Conditional Sum Adder

After the introduction of basic editor operations and specification methods, we will now enter a larger design. By the example of an addition circuit we will show, how the designer can set up short and handy specifications for often used components of larger designs such as microprocessors. He can perform that by using recursive systems of schematic equations.

4.8.1 Basic Concepts of the Conditional Sum Adder

Simple algorithms for addition which e.g. use the well–known carry ripple principle, have the disadvantage of worst (linear) time behaviour. If you need high performance, you have to consider other, more complex structures which will guarantee a better (logarithmic) runtime. We will show now that this is possible by using recursive definitions which will easily mirror the basic principles of the underlying algorithm.

A nice idea for parallelizing addition was given by Sklansky in 1960 ([Skl60]) and is well–known as conditional sum adder. This adder simultaneously computes the sum as well as the sum plus one for the leading parts of the two operands. This is done at the same time as the sum of the lower parts is generated. If the carry of the lower summation is known, you can select the right version of the sum of the upper parts. If this selection can be done in a small amount of time, i.e. by a subcircuit of constant depth, and if we apply this scheme recursively to the upper and lower part of the operands, we get an addition circuit with logrithmic depth, where the operands are split in parts of equal size.

Let $a = num_n(a_{n-1} \dots a_0)$ and $b = num_n(b_{n-1} \dots b_0)$ be two n bit binary numbers. For simplicity we only consider the case $n = 2^k$. Then

$$a_H := num_{\frac{n}{2}}(a_{n-1} \dots a_{\frac{n}{2}}), a_L := num_{\frac{n}{2}}(a_{\frac{n}{2}-1} \dots a_0)$$

denote the $\frac{n}{2}$ bit binary numbers which are represented by the upper (H) and lower (L) half of the operand a. In the same way we define b_H and b_L. Then it holds:

$$\begin{aligned}
a + b &= (2^{\frac{n}{2}} a_H + a_L) + (2^{\frac{n}{2}} b_H + b_L) \\
&= 2^{\frac{n}{2}}(a_H + b_H) + (a_L + b_L) \\
&= 2^{\frac{n}{2}}(a_H + b_H) + 2^{\frac{n}{2}}((a_L + b_L) \div 2^{\frac{n}{2}}) + (a_L + b_L) \bmod 2^{\frac{n}{2}} \\
&= 2^{\frac{n}{2}}(a_H + b_H + (a_L + b_L) \div 2^{\frac{n}{2}}) + (a_L + b_L) \bmod 2^{\frac{n}{2}} \\
&= \begin{cases} 2^{\frac{n}{2}}(a_H + b_H) + (a_L + b_L) \bmod 2^{\frac{n}{2}} & \text{if } (a_L + b_L) \div 2^{\frac{n}{2}} = 0 \\ 2^{\frac{n}{2}}(a_H + b_H + 1) + (a_L + b_L) \bmod 2^{\frac{n}{2}} & \text{if } (a_L + b_L) \div 2^{\frac{n}{2}} = 1 \end{cases}
\end{aligned}$$

and

$$a + b + 1 = \begin{cases} 2^{\frac{n}{2}}(a_H + b_H) + (a_L + b_L + 1) \bmod 2^{\frac{n}{2}} & \text{if } (a_L + b_L + 1) \div 2^{\frac{n}{2}} = 0 \\ 2^{\frac{n}{2}}(a_H + b_H + 1) + (a_L + b_L + 1) \bmod 2^{\frac{n}{2}} & \text{if } (a_L + b_L + 1) \div 2^{\frac{n}{2}} = 1 \end{cases}$$

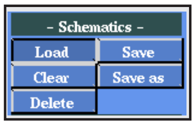
Because $(a_L + b_L) \bmod 2^{\frac{n}{2}}$ (resp. $(a_L + b_L + 1) \bmod 2^{\frac{n}{2}}$) are the $\frac{n}{2}$ lowest bits of the binary representation of the sum $a_L + b_L$ (resp. $a_L + b_L + 1$) and $(a_L + b_L) \div 2^{\frac{n}{2}}$ (resp. $(a_L + b_L + 1) \div 2^{\frac{n}{2}}$) are the carries of $a_L + b_L$ (resp. $a_L + b_L + 1$) we get an arrangement of two $\frac{n}{2}$ bit conditional sum adders. The selection subcircuit for the leading parts of the sum can be constructed by a row of multiplexers which are controlled by the two carry values. This scheme is applied recursively to the two $\frac{n}{2}$ bit adders and leads to a splitting of the operands in very small parts. The most natural input sequence for the whole conditional sum adder therefore is $a_{n-1},b_{n-1},\ldots,a_i,b_i,\ldots,a_0,b_0$. With the same considerations the sequence of the output values of the adder are the bits of the sum plus one and the sum in alternating order $\ldots,(a+b+1)_i$, $(a+b)_i,\ldots,(a+b+1)_0,(a+b)_0$.

Because these input and output sequences are not very intuitive for checking the correctness of our design we will specify two wiring subcircuits (sections 4.8.6,4.8.7, 4.8.8 and 4.8.9) for more convenient input/output behaviour of the conditional sum adder. If you only want to get a first glance at working with the graphical editor you can skip these sections. It is possible to read these sections later in order to complete the specification. The main part of the adder will be specified in sections 4.8.2,4.8.3,4.8.4 and 4.8.5 where the most important functions of the graphical editor are explained in detail.

4.8.2 General Equation for an n Bit Conditional Sum Adder

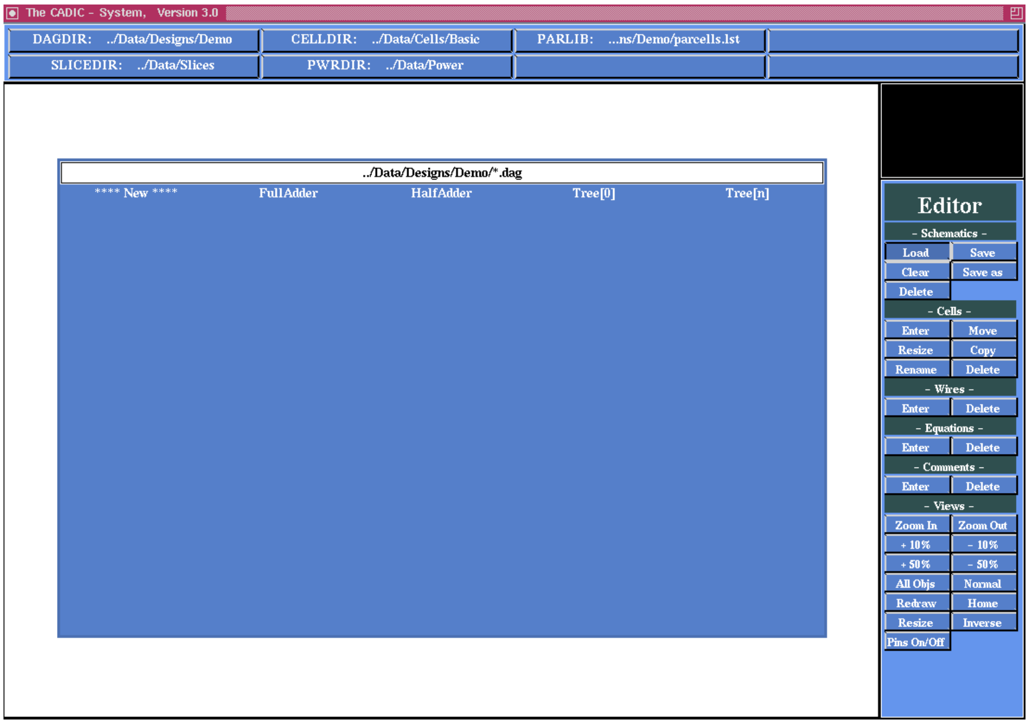
Opening a New Schematic

⇝ 12.1,12.2.1



We will now start with the recursive equation of an n bit conditional sum adder, for which we first have to open a new schematic sheet. This is done by selecting the entry `Load` from the submenu `-Schematics-`. You will see a table of the schematics in the directory given by the environment variable `DAGDIR`.

At this point of the session this table will only contain the entry labelled



The CADIC - System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power

../Data/Designs/Demo/*.dag

**** New ****
FullAdder
HalfAdder
Tree[0]
Tree[n]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

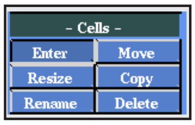
Figure 4.19: Dialog for opening a new schematic

`**** New ****` beside the names of our designs from the previous sections as it is shown in figure 4.19. Select this entry by moving the pointer onto it and press the left mouse button as soon as it is highlighted. After that a text input window will appear on your screen and you have to specify the name of the new schematic. Type in `CSA[n]` and press the return key.

You have opened a new schematic which will first be given by an empty frame. Note, that the name in the upper left field of your environment has changed to `CSA[n]`. The name of the schematic is `CSA`. The square brackets introduce a list of parameters which can be used during the specification of the schematics. These parameters can appear in subcircuits or in the wiring. In our case we have a single parameter `n`, but in general you can use a list of up to 32 parameters.

Entering and Manipulating Cells

⤳ 12.1,12.3.1 In the next step we select the appropriate cells for the recursive specification


- Cells -
Enter
Move
Resize
Copy
Rename
Delete

of the conditional sum adder and place them within the given frame. For this purpose select the entry `Enter` from the submenu `-Cells-` in the menuline. After picking this point with the left mouse button, you will see three new windows upon your working area. Each of these windows contains a list of cells of a certain type (c.f. figure 4.20).

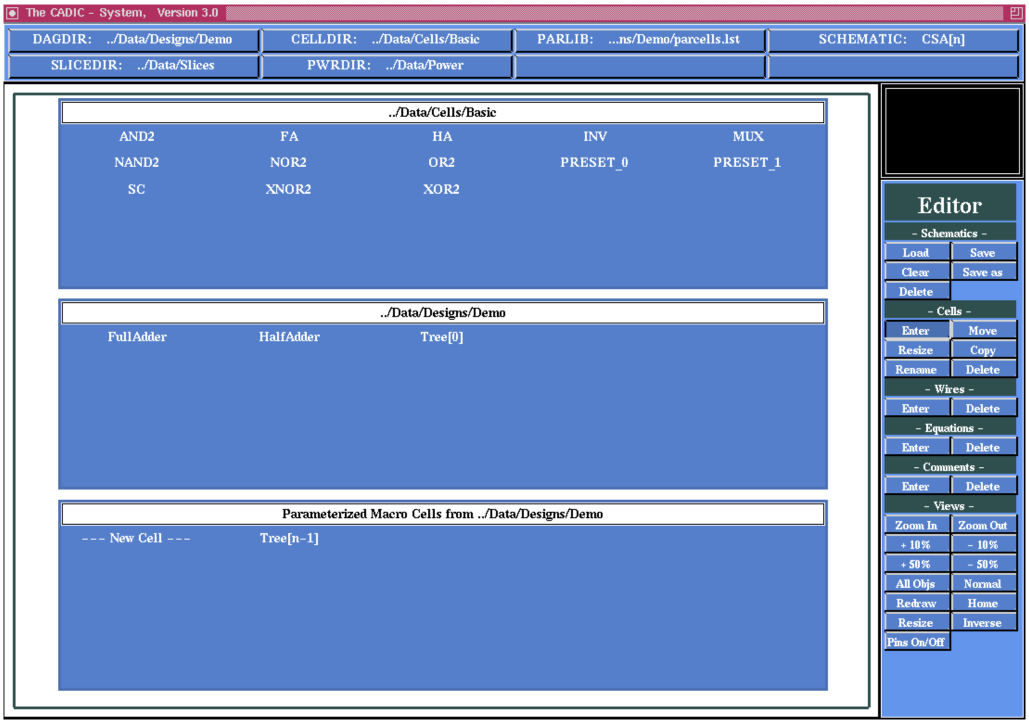

The CADIC - System, Version 3.0
DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ../Data/Slices
PWRDIR: ../Data/Power
../Data/Cells/Basic
AND2
FA
HA
INV
MUX
NAND2
NOR2
OR2
PRESET_0
PRESET_1
SC
XNOR2
XOR2
../Data/Designs/Demo
FullAdder
HalfAdder
Tree[0]
Parameterized Macro Cells from ../Data/Designs/Demo
--- New Cell ---
Tree[n-1]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse
Pins On/Off

Figure 4.20: Dialogs for cell and macro selection

In the upmost window you see the names of the cells in the basic cell library which are located in the directory described by the environment variable `CELLDIR`. The second window shows the names of macro cells which have been defined by a previous input of a schematic. The system only shows such macros that are fixed, i.e. which do not depend on any formal parameters. This is the reason, why the macro `Tree[0]` appears in this list and the parameterized macro `Tree[n]` does not. In the third window the names of the parameterized macro cells are listed which have been used during the current session. Actually this list contains the entry labelled `--- New Cell ---` beside the previously specified parameterized macro `Tree[n-1]`. Select the first one in the same way as you have selected to open a new schematic in order to define a new parameterized macro cell. After hitting the left mouse button you will be prompted to enter a new macro name. Type in `CSA[n/2]` to specify a macro for a conditional sum adder for $\frac{n}{2}$ bit operands.

Note that you can only use the parameters of the schematic to build arithmetic expressions for the parameters of the macro cells. Try to specify a cellname `CSA[k/2]` and the system will respond with the message

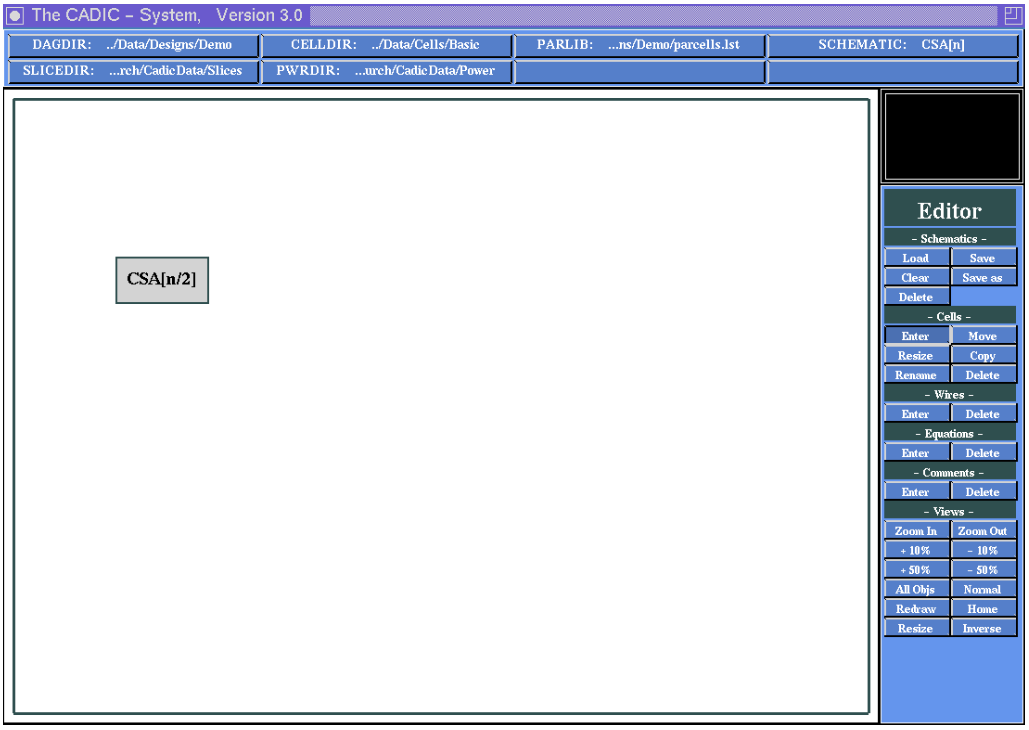
```
!  Error !  "CSA[k/2]" uses illegal parameter "k"
```

The syntax diagrams for the expressions allowed as parameters of these macro cells are given in chapter 12.

After the specification of a correct cellname you will see the graphical representation of the macro, i.e. a red bounded box with the cellname in its center. You can move this symbol inside the frame and place it at the desired position by clicking the left mouse button. The macro can only be placed on legal positions, i.e. there must not be an overlapping with other macros, wires or the border of the schematic. Try to move the macro onto the schematic frame and you will notice a change of its colour from red to grey. This change indicates the selection of an illegal position. Put the macro onto the position you see in figure 4.21.

The size of the macro is chosen by default, because the selected parameterized macro represents a whole class of macros. Its exact size can be determined, if the values for the parameters are known. For the abstract specification level of the editor the size of a cell or a macro is not important. In order to setup a comfortable schematic which is easy to understand, we will now resize the macro. This gives an impression of the relative sizes of the macros. Before we can resize the macro, we have to terminate the enter cell function. After you have placed the macro the three cell selection windows are popped up again in order to select another cell or macro. You can abort this function by pressing the right mouse button within one of the three windows.

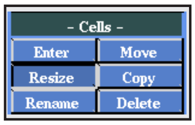
↝ 12.3.3 For the resize operation push the button `Resize` in the `-Cells-` submenu. Then you are in the cell selection mode and can pick the desired cell within


The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

CSA[n/2]
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

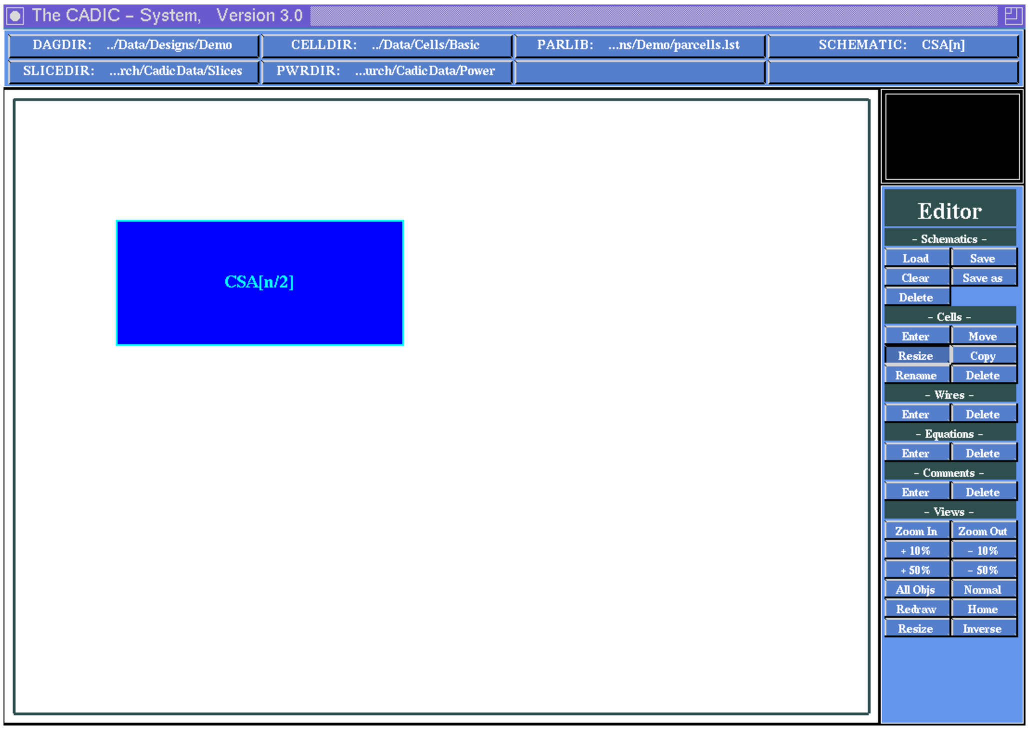
Figure 4.21: Placing a parameterized macro cell for the recursive specification of the conditional sum adder


– Cells –
Enter
Move
Resize
Copy
Rename
Delete

the workarea. Move the pointer onto the workarea and you will see the macro being highlightet (its colour changes from grey to blue). In cell selection mode always the macro nearest to the pointer will be highlighted, you need not be inside it. At this point of our session cell selection is trivial because we only have a single macro within the workarea. The selection is done by clicking the left mouse button.

After the selection you can draw a rubberbanding frame which is anchored in the upper left corner of the macro and represents its new size. During the resize operation you need not hold down the left button. Just move the pointer to the desired position and confirm the new size by clicking the left mouse button or cancel the operation with the right mouse button (you will be returned to cell selection mode).

If you confirm a new size for the macro you will be in cell placement mode. The reason for this are possible overlappings of the resized macro with its



The CADIC - System, Version 3.0

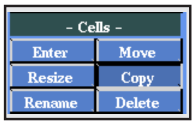
DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

CSA[n/2]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.22: Resizing a parameterized macro cell

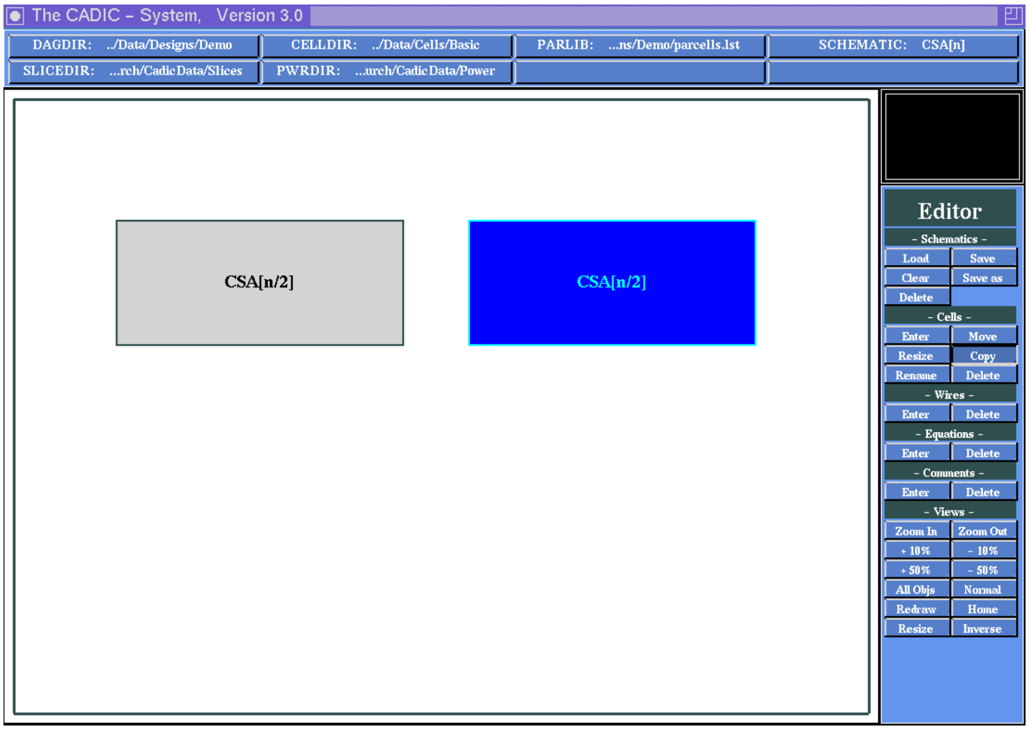
neighbours or with wires. You have to choose a new position for it which is only possible within the given constraints (i.e. no overlapping with macros, wires or the border). Move the resized macro to the previous position and confirm with the left mouse button. Now your schematic should look like that shown in figure 4.22.

⇝ 12.3.4



- Cells -
Enter
Move
Resize
Copy
Rename
Delete

As described above the conditional sum adder simultaneously generates the sum and sum plus one for the upper and lower halfs of the operands. We now have positioned the macro `CSA[n/2]` for the upper half of the operands. A second instance of this macro will be needed for the lower half. We could create this instance by following the above steps (selecting a parameterized macro, placing and resizing it). But the editor offers a much simplier way to perform this operation. Push the entry `Copy` from the `-Cells-` submenu. This enables the cell selection mode where you can select an instance to be doubled. Press the left mouse button and you will get a second macro labelled `CSA[n/2]` which has the same size as our first macro.


The CADIC – System, Version 3.0

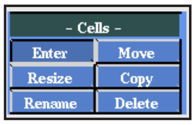
DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

CSA[n/2]
CSA[n/2]
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.23: Creating a copy of an exisiting macro cell

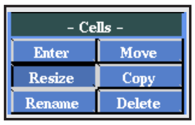
Place it within the workarea in order to have the schematic look like that in figure 4.23. During the placement operation you can watch the macro changing its colour, if it is placed on the first `CSA[n/2]` or on the border. In this case you cannot drop it, i.e. pressing the left mouse button has no effect. You can cancel the operation by pressing the right mouse button. The copy will disappear and you will return to cell selection mode. Pressing the right mouse button again will terminate the whole copy operation.

↝ 12.3.1

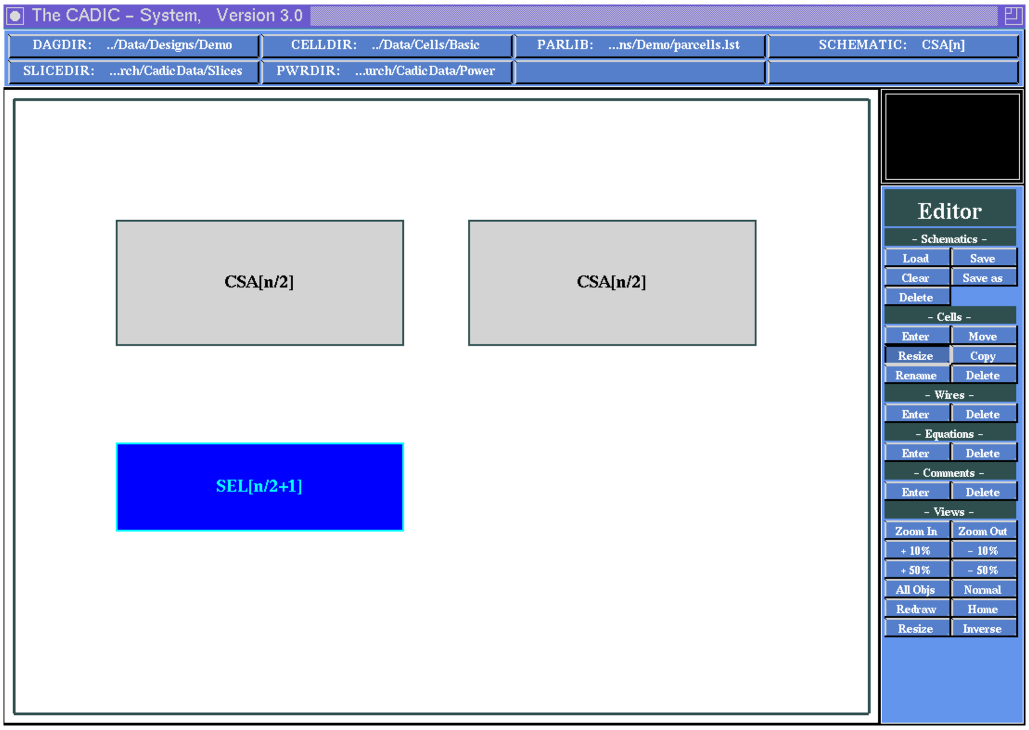

– Cells –
Enter
Move
Resize
Copy
Rename
Delete

In the next step we create an instance for the subcircuit which selects the sum or the sum plus one from the conditional sum adder for the upper halfs. This selection depends on the carry values of the addition of the lower halfs of the operands. This subcircuit will be represented by a new parameterized macro labelled `SEL[n/2+1]`. Create it in the same way as you have created the first `CSA[n/2]` macro and place it below the left $\frac{n}{2}$ bit conditional sum adder as it is shown in figure 4.24 (in this figure the macro is also resized as it will be described in the next paragraph).

↝ 12.3.3


- Cells -
Enter
Move
Resize
Copy
Rename
Delete

The width of this macro should be the same as that of `CSA[n/2]` which we achieve by an appropriate resize operation. The height of this macro should be smaller than the height of `CSA[n/2]` because it only will contain a line of multiplexers (the selection subcircuit must have constant depth as mentioned above). Now we have selected, resized and positioned the macros needed for the general recursive equation of the conditional sum adder. It is a recursive description because we need two instances of a macro which has the same name as the whole schematic (`CSA[n]` uses `CSA[n/2]`). Before we start to draw the wires between the macros, the schematic should look like that in figure 4.24.


The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

CSA[n/2]
CSA[n/2]
SEL[n/2+1]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.24: Parameterized macro cells for the n bit conditional sum adder

Entering Wires

↝ 12.4.1

- Wires -
Enter
Delete

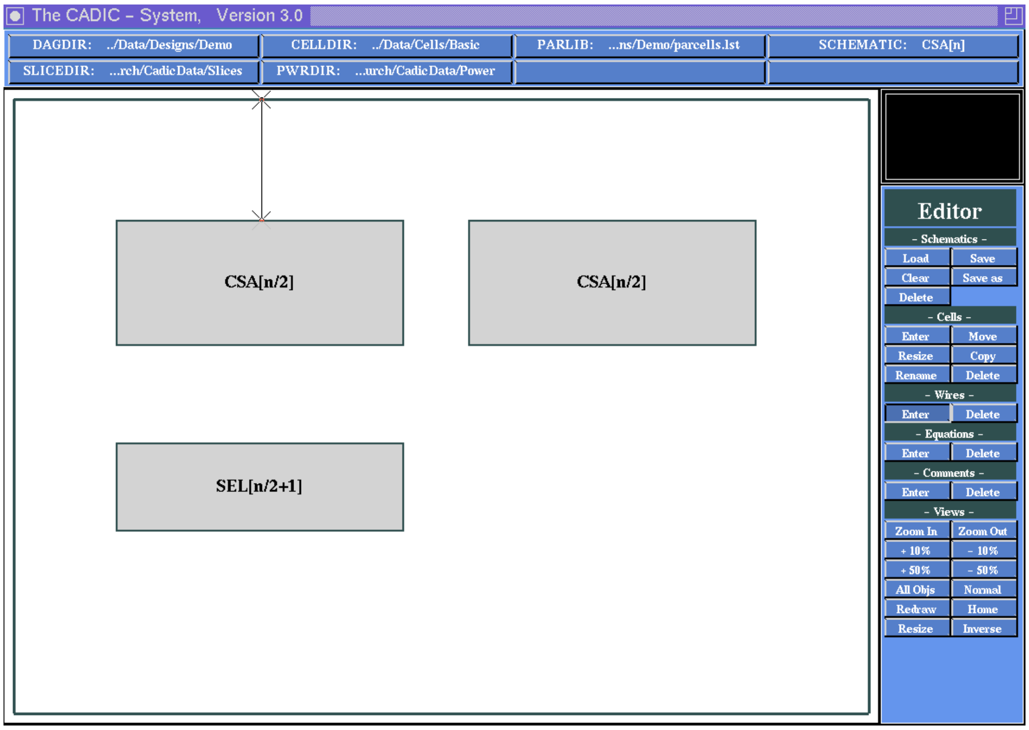
In this section we show how to connect the modules with wires. The main point you will observe during this procedure is that because of the parameterized description the lines drawn will have variable widths. The width of a line specifies the number of parallel wires contained in this bundle. We start with the left `CSA[n/2]` which computes the sum and sum plus one of the upper half of the operands. First select the point `Enter` from the `-Wires-` submenu and move the pointer into the workarea. The cursor has changed to a crosshair and you can select the start point for a new wire. Move the pointer to the northern side of the left `CSA[n/2]` macro and position it to the middle of this side. If you move the pointer slowly toward the border of the macro you can observe that it snaps onto the border, if the distance becomes less than a certain tolerance value. This indicates that you have selected a special position within the schematic. The same behaviour can be noticed if you reach the frame, a pin of a cell or another wire. If the pointer has snapped to the border of the macro press the left mouse button to fix the start point of the new wire. Now move the pointer vertically towards the northern border of the schematic. You will see that it is followed by a grey rubberband line which indicates the position of the new wire. The cursor snaps to the border if the distance is small enough (your screen should look like that shown in figure 4.25).

Now you can fix the end point of the wire by pressing the left mouse button again. The grey rubberband line will disappear and you will be prompted to enter the width of this wire. As mentioned above the width of a wire represents the number of parallel lines within this bundle. In our case this wire describes the input line to the first conditional sum adder for the upper half of the operands. The width of a single operand is given by the parameter n. The left `CSA[n]` gets two operands of length $\frac{n}{2}$, i.e. the number of single wires is n. We describe this wire by a variable `@high[n]` which implicitly represents the two upper halfs of the operands on each stage of the recursion.

```
Please enter Wire Width:  <@high[n]      >
```

Type in `@high[n]` and confirm with the return key. You will see the new wire as a yellow line from the border of the `CSA[n]` to the border of the schematic. In the middle of the wire you can notice the description of its width annotated by a small diagonal line with the appropriate expression (c.f. figure 4.26).

Note that you can only use the parameters of the schematic to build arith-



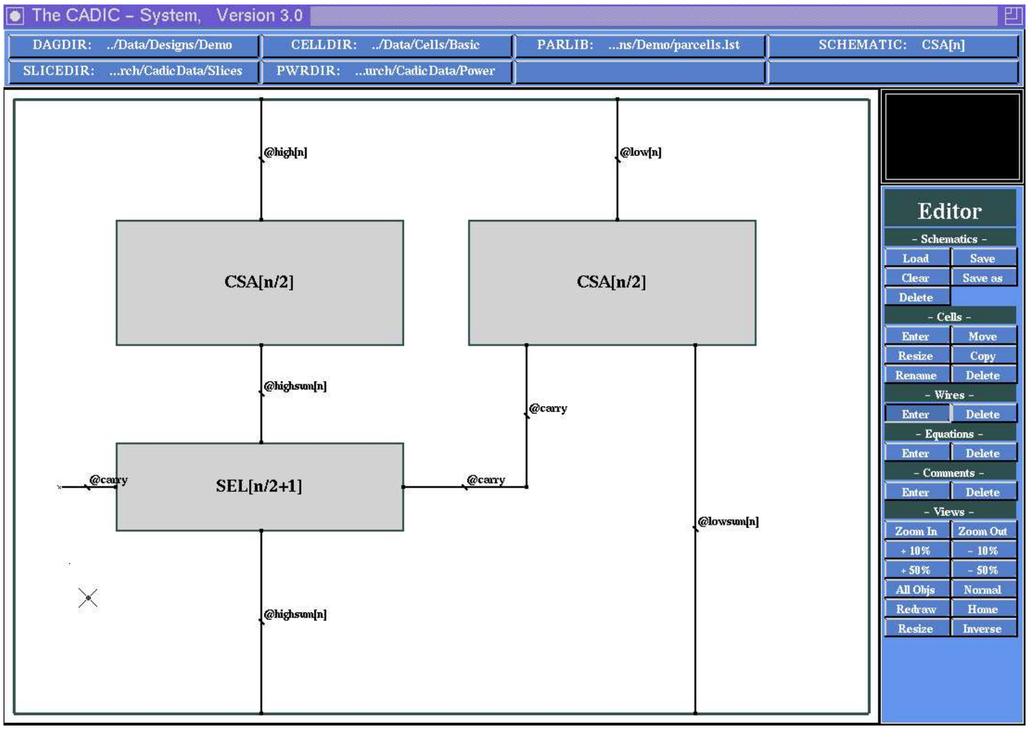
DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

The CADIC – System, Version 3.0
CSA[n/2]
CSA[n/2]
SEL[n/2+1]
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.25: Connecting a module with the northern border of the schematic

metic expressions or wire variables. The exact syntax for these expressions is given by the diagrams in appendix.

We proceed with the other modules. The right instance of `CSA[n/2]` will also be connected with the northern border of the schematic. You can draw this wire in the same way as you did for the left `CSA[n/2]`. Type `@low[n]` for its width in order to denote that it describes the wires for the lower parts of the operands. Now we turn to the southern borders of the two `CSA[n/2]`. The left one must be connected with the northern border of the `SEL[n/2+1]`. Its width should describe the sum and sum plus one of the upper parts of the operands. Therefore we call it `@highsum[n]`. In this name we also have encoded the two carry bits for the higher part. A second wire with the width `@highsum[n]` has to be drawn from the southern border of `SEL[n/2+1]` to the southern border of the schematic. In the lower part we split the carry wires from the rest of the sums because we need the carries to control the selection subcircuit. First you draw a line from the southern border of the right `CSA[n/2]` to the southern border of the schematic and call it `@lowsum[n]`. Next you draw a line which starts at the southern border of the right `CSA[n/2]` left of the line `@lowsum[n]` to the eastern side of the selection subcircuit `SEL[n/2+1]`. The width of this line will be denoted by `@carry` which describes the carry bits of the sum and sum plus one for the lower parts of the operands. You can directly connect the southern and the eastern side of the two macros. Note that the rubberbanding line will follow the movement of the pointer and there will be generated a knee when you change the motion direction from south to west.



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

@high[n]
@low[n]
CSA[n/2]
CSA[n/2]
@highsum[n]
@carry
@carry
SEL[n/2+1]
@carry
@lowsum[n]
@highsum[n]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.26: Final wiring for the conditional sum adder

The last wire we have to draw in this schematic starts at the western border of the selection subcircuit. It represents the carry lines from the `CSA[n/2]` for the lower halfs of the operands which are routed through the selection subcircuit. Because of the recursive specification of this subcircuit we have to connect a wire to its western border (c.f. section 4.8.4). This wire must not be connected to the western border of the schematic because of the recursive definition of the conditional sum adder. This means that we have to cut the wire within the schematic. This is done by a so called projection cell which is a legal operation in the underlying calculus (c.f. section 4.2). The graphical notation for a projection cell is very simple. Just place the endpoint of the wire anywhere within the schematic and press the left mouse button. In our case we draw a small horizontal line towards the western border of the schematic as you can see it in figure 4.26.

To leave the enter wire mode press the right mouse button twice. The first button press will deactivate the selection of a wire end point and return to the selection of new start point. The second button press will abort the enter wire mode and return to the menu selection mode.

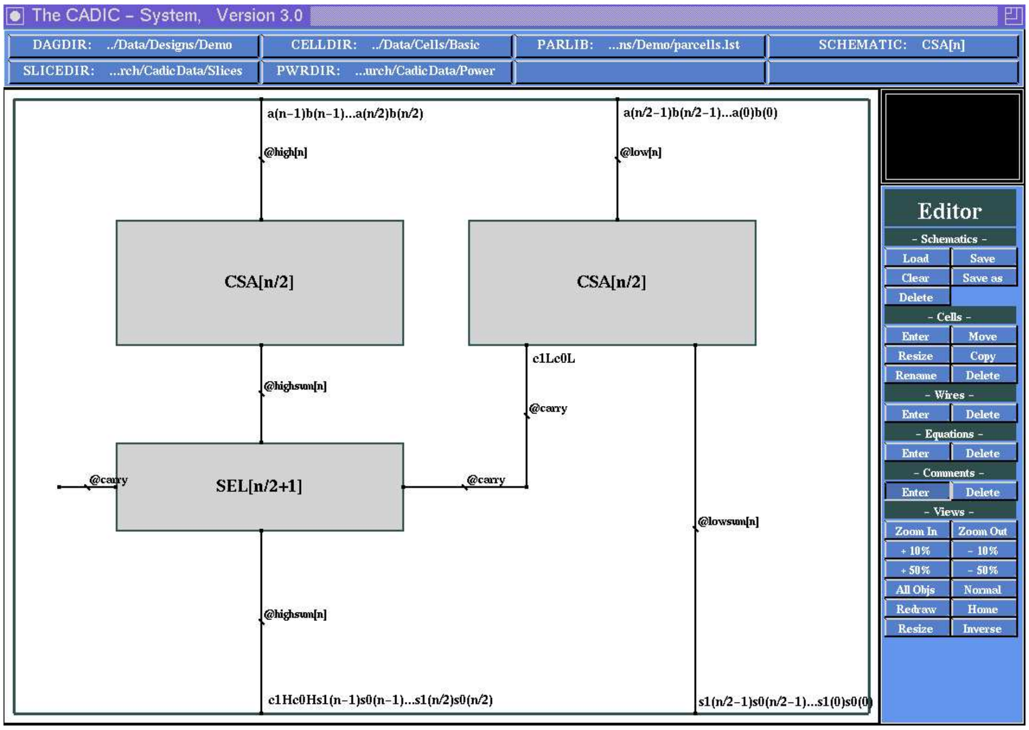
Entering Comments

↝ 12.5.1

- Comments -
Enter
Delete

In order to document the schematic and especially the signals of the wiring you can place remarks at any position within the drawing. To do this select the entry `Enter` from the `-Comments-` submenu. Then you are prompted to enter some comment text. First we label the input wires of the schematic. For the left wire type in `a(n-1)b(n-1)...a(n/2)b(n/2)` and confirm this by pressing the return key. The comment text is now visible within the workarea and it follows the movement of the pointer. Move it near to the left input wire and fix its position by pressing the left mouse button. This causes the input window to popup again in order to enter another comment text. For the right input wire of the schematic type in `a(n/2-1)b(n/2-1)...a(0)b(0)` and place it at the appropriate position (c.f. figure 4.27).

Now we turn to the output signals of the schematic. The left wire, connected with the southern border of the schematic, will be labelled with the comment `c1Hc0Hs1(n-1)s0(n-1)...s1(n/2)s0(n/2)`. The right wire at the southern border is labelled with `s1(n-1)s0(n-1)...s1(0)s0(0)`. Finally the carry wire is marked with the comment text `c1Lc0L`. To terminate the enter comment mode simply press the return key within an empty input window. Your schematic should now look like that in figure 4.27.



The CADIC - System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

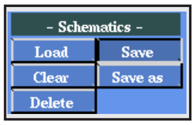
a(n−1)b(n−1)...a(n/2)b(n/2)
a(n/2−1)b(n/2−1)...a(0)b(0)
@high[n]
@low[n]
CSA[n/2]
CSA[n/2]
c1Lc0L
@highsum[n]
@carry
@carry
SEL[n/2+1]
@carry
@lowsum[n]
@highsum[n]
c1Hc0Hs1(n−1)s0(n−1)...s1(n/2)s0(n/2)
s1(n/2−1)s0(n/2−1)...s1(0)s0(0)
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.27: Comments placed at the input and output wires of `CSA[n]`

Note that the comment text are used to document the schematics. If you use the parameters of the schematic, the comment text will not be evaluated, if you fix the parameter values as it is shown in section 5.

Saving a Schematic

↝ 12.2.2



- Schematics -
Load
Save
Clear
Save as
Delete

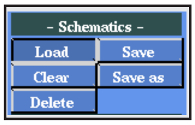
Now that we have drawn all elements of the general recursive equation for the n bit conditional sum adder we must save the schematic. This is simply done by selecting the entry `Save` from the `-Schematics-` submenu. The schematic will be saved to the file `CSA[n].dag` in the directory given by the environment variable `DAGDIR`.

4.8.3 Basic Equation for a 1 Bit Conditional Sum Adder

Opening a New Schematic

↝ 12.2.1

After the specification of the general recursive equation for the conditional



sum adder we will now draw the schematic for the basic equation. This will be an adder for 1 bit operands. Therefore we call this schematic `CSA[1]` which will be opened following the steps from above. Note that the list of schematics now contains the entry `CSA[n]` beside the position of `**** New ****`. After selecting a new schematic you should type in the name `CSA[1]` and press return. Now we have created an empty frame for entering the 1 bit conditional sum adder.

Entering Basic Cells

The task of this adder is to compute the sum and the sum plus one of two single bits a_0 and b_0. The following table shows the values of this function:

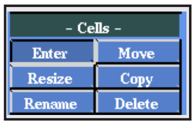
a_0	b_0	s_1^H	s_1^L	s_0^H	s_0^L
0	0	0	1	0	0
0	1	1	0	0	1
1	0	1	0	0	1
1	1	1	1	1	0

With $s_1 = (s_1^H, s_1^L) = a_0 + b_0 + 1$ and $s_0 = (s_0^H, s_0^L) = a_0 + b_0$ we can derive following boolean functions from this table

$$s_0^H = a_0 \wedge b_0, s_0^L = a_0 \oplus b_0, s_1^H = a_0 \vee b_0, s_1^L = \overline{s_0^L}.$$

We have to draw a schematic with two input signals (a_0, b_0) and four ouput signals $(s_1^H, s_0^H, s_1^L, s_0^L)$. The sequence of the output signals must have the given form because of the specification of the general equation for the conditional sum adder.

⇝ 12.3.1

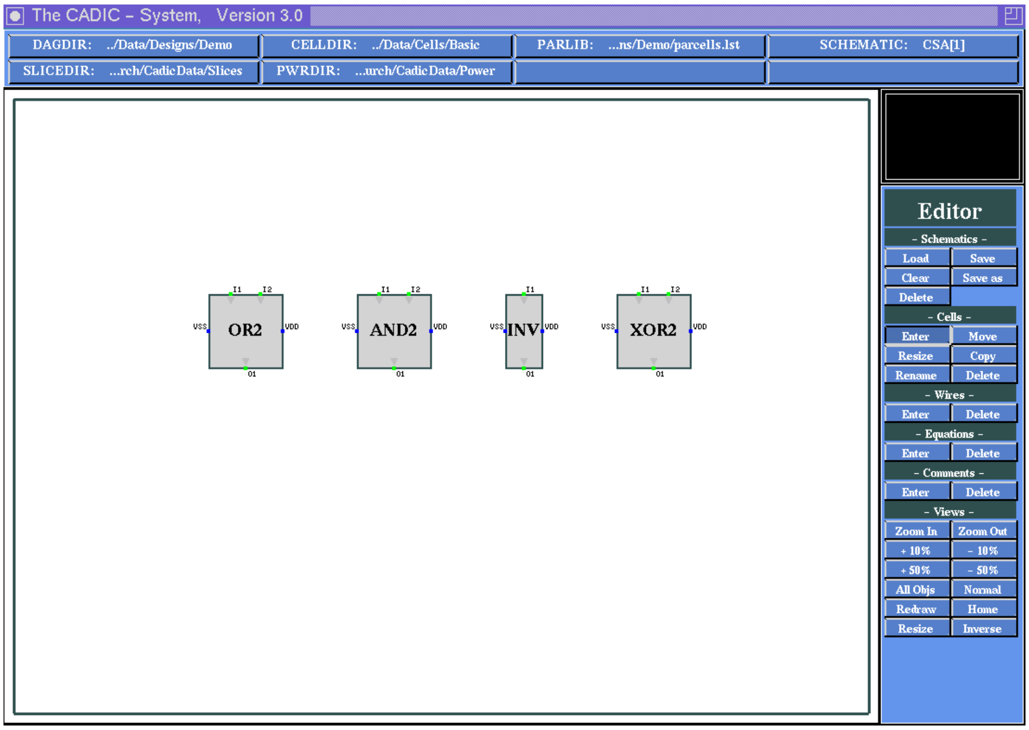


First we select and place the needed basic cells. To do this select the entry `Enter` from the submenu `-Cells-`. This will open the three cell list windows (c.f. figure 4.7). Now we have to select the cells from the upmost window which contains the names of the available basic cells. These cells are located in the directory given by the environment variable `CELLDIR`.

The first cell we need is an OR gate with two inputs. Move the pointer to the name `OR2` and press the left mouse button if the name is highlighted.

The cell list windows will be closed and you see the shape of an OR gate in the workarea. The gate will follow the pointer motion within the workarea and you can pick the right position to place it. As shown above we need four gates to compute the output values of `CSA[1]`. The OR gate will be the leftmost (c.f. figure 4.28).

If you drop the gate by pressing the left mouse button, it is redrawn with two input pins at its northern border and an output pin at its southern border. The pins are labelled with the names `I1`, `I2` and `O1`. Small arrows indicate the directions of the pins.



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[1]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

I1
I2
VSS
OR2
VDD
O1
I1
I2
VSS
AND2
VDD
O1
I1
VSS
INV
VDD
O1
I1
I2
VSS
XOR2
VDD
O1
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.28: Placement of the basic cells for the 1 bit conditional sum adder

After the placement the cell list windows are popped up again in order to select another gate. Move the pointer to the entry `AND2` to get an AND gate with two inputs. We need this gate to compute the second value s_0^H, so place it right beside the OR gate. In the same way you should select an Inverter (`INV`) and an EXOR gate for two inputs (`XOR2`) and place them according to figure 4.28). After the placement of the last gate you can abort the cell selection mode by pressing the right mouse button.

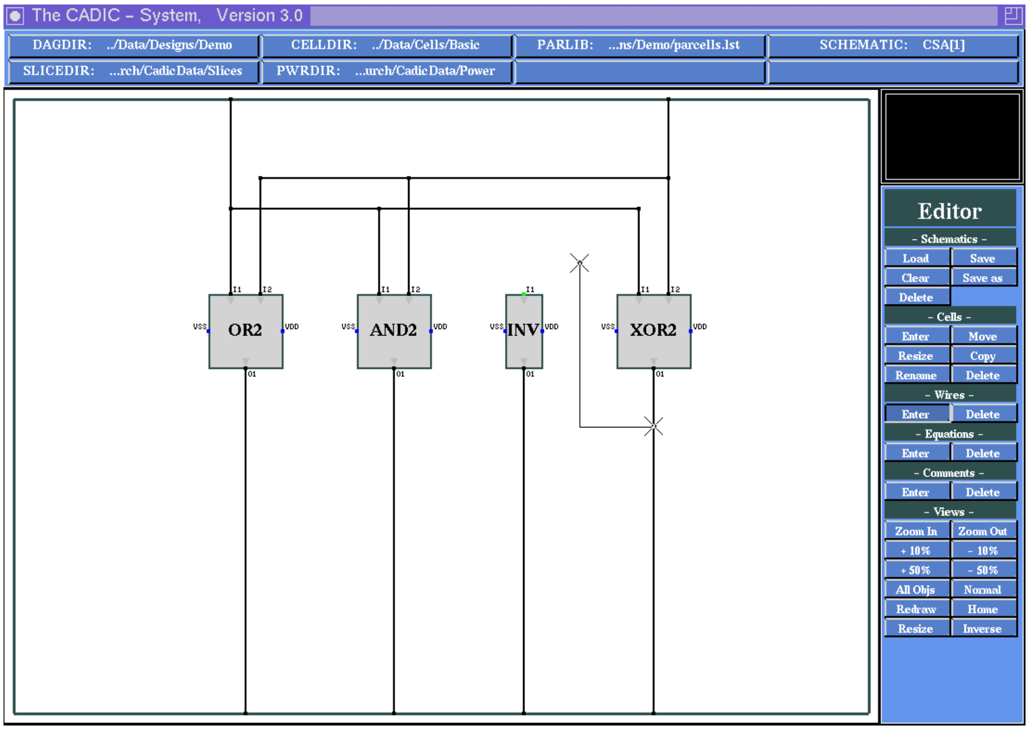
Entering Wires

⤳ 12.4.1

- Wires -
Enter
Delete

To draw the wiring of `CSA[1]` select the entry `Enter` from the `-Wires-` submenu. First we connect the output pins of the gates directly with the southern border of the schematic. Move the pointer near to the output pin of the `OR2` gate. If the distance to the pin is less than a certain tolerance value the crosshair will snap to the pin and you can fix the start point of the wire by pressing the left mouse button. Move vertically to the southern border of the schematic and fix the end point in the same way.

When you drew the wires in the general equation `CSA[n]` you have been prompted to enter the width of the wire (e.g. `@carry`). In the case of connecting a pin of a basic cell this is not necessary because the wire represents a single binary value. The width is automatically set to 1.



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[1]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

I1
I2
VSS
VDD
OR2
AND2
INV
XOR2
O1
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.29: Wiring for the 1 bit conditional sum adder

After connecting the last three output pins of the gates with the southern border of the schematic we turn to the input pins. Connect the left input (`I1`) of the `OR2` gate as well as the right input (`I2`) of the `XOR2` gate directly with the northern border of the schematic. These two wires represent the inputs a_0 and b_0 of the 1 bit adder. As shown above we have to feed these values into the `OR2`, `AND2` and `XOR2` gate. To do this we branch the wires and connect them with the appropriate input pins. To do this move the pointer onto the left wire. You can notice that it snaps to the wire if the distance is less than a certain tolerance value. Press the left mouse button to fix the start point and move the pointer to the left input pin of the `XOR2` gate. The connection is indicated by a rubberbanding line between the start point and the current position of the pointer. If you select an illegal position the rubberbanding line will disappear. This is the case if the chosen connection intersects a cell or moves along an already existing wire or the border of the schematic (try this by moving the pointer into the `AND2` gate). If you have setup the right connection to the left input of the `XOR2` gate press the left mouse button to fix the end point of the wire. The yellow dot at the start point of the new wire indicates a branching of the previous vertical wire. Implicitly the branch node is considered to be a basic operation in the underlying calculus just as the projection node mentioned above. In this special case we have entered an east–branch.

In the same way you can connect the right vertical wire with the right input pin of the `OR2` gate. Both new wires still have to be connected with the input pins of the `AND2` gate. This can be done by two simple vertical wires. Here it is important that you select the input pin of the `AND2` gate as the start point and move the pointer vertically toward the appropriate wire creating a new branch (south–branch). If you fix the start point on the horizontal wire you might miss the right position for a vertical connection. The position of the pin implies the position of the wire, so selecting it first is often very helpful. Now complete the wiring of `CSA[1]` by connecting the wire from the output of the `XOR2` gate with the input pin of the Inverter. As shown above the `XOR2` gate computes the value s_0^L and the value $s_1^L = \overline{s_0^L}$ is just the inverted signal. This connection has to be entered in two steps. First fix the start point of the wire at the output wire of the `XOR2` gate and move the pointer upwards between the `INV` and the `XOR2` as it is indicated in figure 4.29. At this position fix the end point of this wire segment. From this end point move the pointer to the pin connector of the input of `INV`. The previous end point serves as the start point for the next wire segment which you can fix now by pressing the left mouse button. You should have entered a schematic similar to that shown in figure 4.30. Terminate the enter wire mode by pressing the right mouse button two times.

Entering Comments

⇝ 12.5.1

- Comments -
Enter
Delete

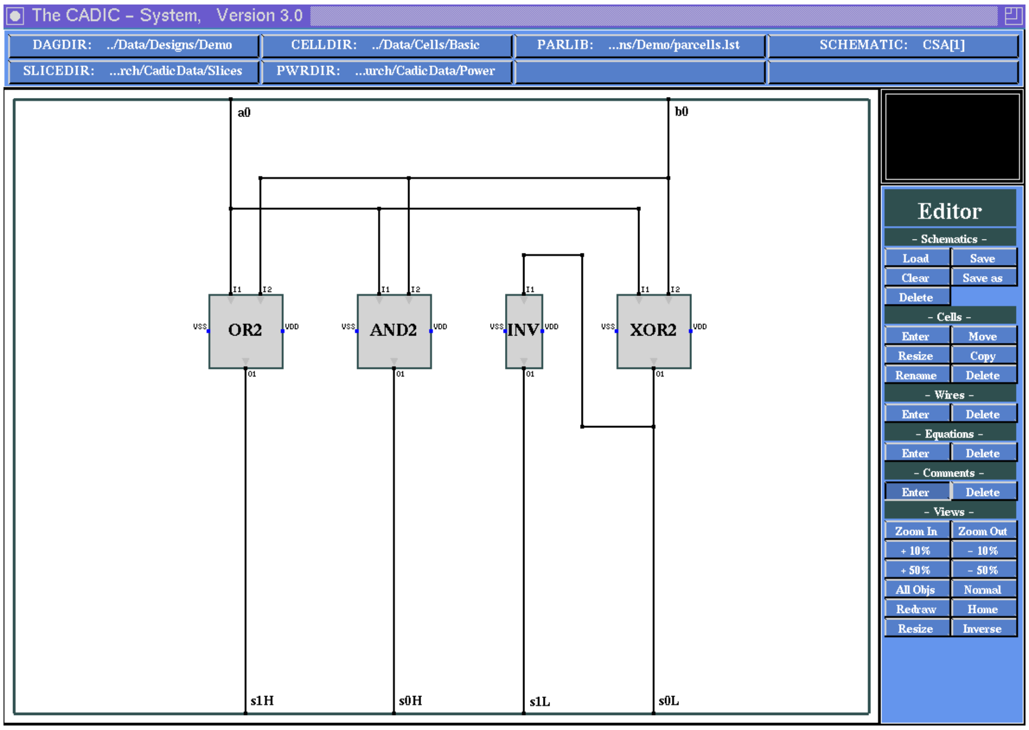
Before we save the schematic we want to document it by some comments in order to later remember the meaning of the different signals. For this purpose select the entry `Enter` from the `-Comments-` submenu. Now you are prompted to enter the comment text. First we will describe the input signals of the schematic. The left one represents the value a_0, so type in `a0` and press the return key. You will see the comment text in the upper left corner of the workarea. The text is connected to the pointer and will follow its movement within the workarea. Move it near to the connection of the left input signal with the northern border of the schematic and place it on the right side of the wire by pressing the left mouse button. After the placement of the comment the text field window is popped up again and you can enter the next comment text. For the second input wire type in `b0` and move it to the analogous position.

Now we turn to the output signals of `CSA[1]`. The sequence from left to right is $(s_1^H, s_0^H, s_1^L, s_0^L)$. That is why we place the remaining four comments `s1H`, `s0H`, `s1L` and `s0L` near to the points where the wires are connected to the southern border of the schematic. After the placement of the last comment you can abort the loop of entering a comment text by simply pressing the return key to an empty input window. The schematic `CSA[1]` should now look like that shown in figure 4.30.

Saving the Schematic

⇝ 12.2.2

Save the schematic by selecting the entry `Save` from the `-Schematics-`

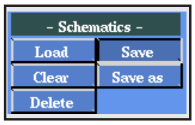


The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSA[1]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

a0
b0
I1
I2
VSS
OR2
VDD
O1
AND2
INV
XOR2
s1H
s0H
s1L
s0L
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.30: Remarks for the input and output signals of `CSA[1]`



- Schematics -
Load
Save
Clear
Save as
Delete

submenu. The graphical input is then written to the file `CSA[1].dag` in the directory given by the environment variable `DAGDIR`.

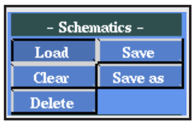
4.8.4 General Equation for the Selection Subcircuit

At this point of the session we have entered the general and the basic equation for the recursive description of an n bit conditional sum adder. In the general equation `CSA[n]` we used a parameterized subcircuit `SEL[n/2+1]` for the selection of either the sum or the sum plus one of the upper halfs of the operands. This selection circuit is controlled by the values of the carry bits of the sum of the lower half. In order to generate an adder with logarithmic depth the selection circuit must have constant depth. In the general equation of the conditional sum adder we used an instance of the selection subcircuit in advance, i.e. before we have defined this circuit.

We will construct this subcircuit by a row of multiplexers. The generation of a row of k elements is a basic recursive technique which is used in a lot of parameterized designs. If the number of elements to be generated is a power of two the recursion is even more simple than in our case where we generate an arbitrary number k ($k > 0$) of elements. We need two schematic inputs for the specification, one for a general and one for a basic equation. The principle is a simple divide and conquer technique, i.e. a row of k elements consists of a row of $\lceil \frac{k}{2} \rceil$ elements followed by a row of $\lfloor \frac{k}{2} \rfloor$ elements. Here we use the reduction $k \longrightarrow \lceil \frac{k}{2} \rceil + \lfloor \frac{k}{2} \rfloor$ which leads to a structure of logarithmic size as we will show it in chapter 5. You could also choose the form $k \longrightarrow (k-1)+1$ which would result in a structure of linear size.

Opening a New Schematic

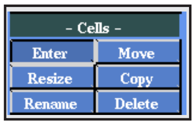
⇝ 12.2.1



An important feature of the graphical editor of the CADIC system is that such theoretical considerations can directly be used to set up an appropriate schematic. The general equation for the row of k multiplexers is called `SEL[k]`. You should open a new schematic now (select the entry `Load` from the `-Schematics-` submenu and `**** New ****` in the schematic list, then type in the appropriate name).

Entering and Manipulating Parameterized Macros

⇝ 12.3.1

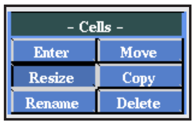


In the empty frame we first place a new parameterized macro cell. Select the entry `Enter` from the `-Cells-` submenu to popup the cell selection windows. In the third window which contains the names of parameterized macro cells, move the pointer onto the `--- New Cell ---` entry and press the left mouse button if it is highlighted. In the following input line type `SEL[upper(k/2)]` for the name of the new macro cell. The function call `upper(k/2)` computes the value $\lceil \frac{k}{2} \rceil$ (for more information about standard function calls see section 12.1). You get a new parameterized macro with the default size. This macro can be moved around within the workarea. Place it near to the western border of the schematic and press the left mouse button. After that the cell selection windows are popped up again. We will generate the second instance from this first one, so you can abort

⇝ 12.1

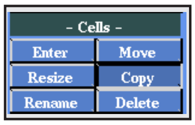
the cell selection mode by pressing the right mouse button within one of the three list windows.

⇝ 12.3.3



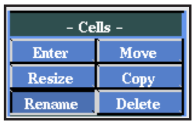
Before we create a copy of this instance we resize it to a convenient size. Select the entry `Resize` from the `-Cells-` submenu. You are in cell selection mode when you move the pointer into the workarea. Notice that the only instance `SEL[upper(k/2)]` is highlighted in blue. Select it by pressing the left mouse button. After that you can resize it in the same way as we did during the specification of `CSA[n]` for the first instance `CSA[n/2]` (c.f. page 62). The pointer motion is coupled with the drawing of a rubberbanding rectangle which indicates the new size of the macro cell. After confirming the new size with the left mouse button you are in the cell placement mode. Move the macro to its previous position and press the left mouse button again. You are returned to the cell selection mode for further resize operations. Abort this mode by pressing the right mouse button within the workarea.

⇝ 12.3.4



Now we create a copy of this instance for the second part which contains the last $\lfloor\frac{k}{2}\rfloor$ elements. If you wonder why we take an instance with the same name, you will later see that its label can be changed to the correct value. Select the entry `Copy` from the `-Cells-` submenu and move the pointer into the workarea to choose the instance `SEL[upper(k/2)]`. If you press the left mouse button you get a second instance of this name with the same size like the first one. Now the cell placement mode is active and you can move the new instance right beside the first `SEL[upper(k/2)]`. Fix its position by pressing the left mouse button, then abort the cell selection mode by clicking with the right mouse button.

⇝ 12.3.5



⇝ 12.1

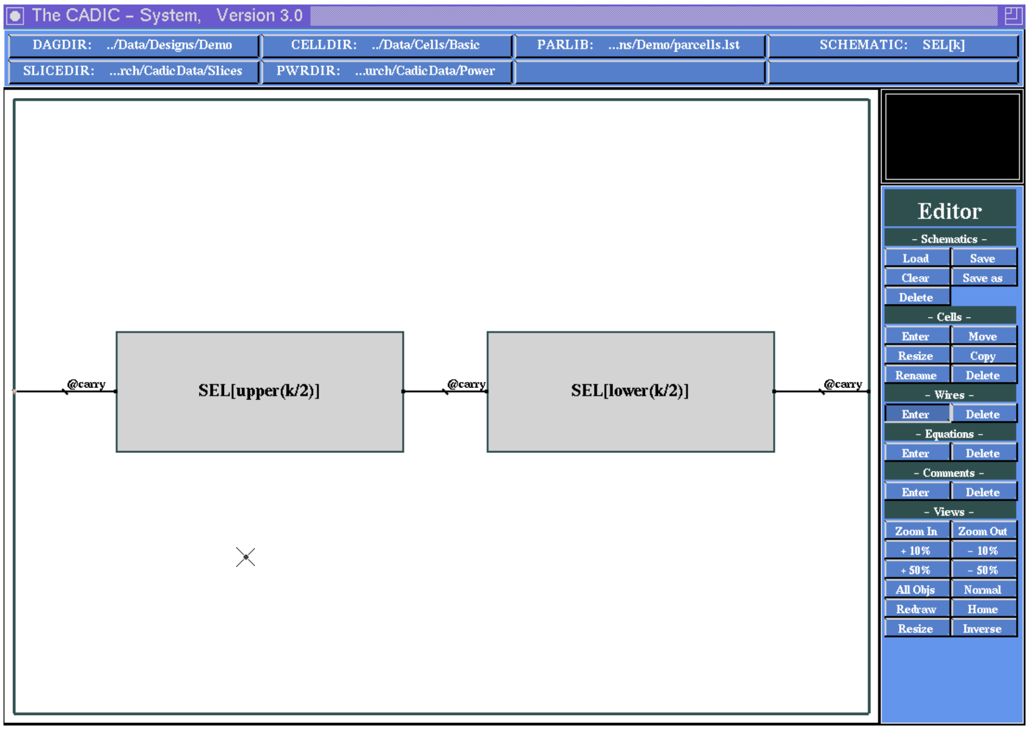
As shown above we will recursively construct a row of k elements by the sequence of $\lceil\frac{k}{2}\rceil$ and $\lfloor\frac{k}{2}\rfloor$ elements. For the second instance we have to change its parameter from `upper(k/2)` to `lower(k/2)` (the function call `lower(x)` computes the value $\lfloor x\rfloor$, c.f. section 12.1). Now we change the name of the instance in this way. For this purpose select the entry `Rename` from the `-Cells-` submenu. Select the right `SEL[upper(k/2)]` by moving the pointer near to it and press the left mouse button when the instance is highlighted. The text input window will popup on the screen and you can type in `SEL[lower(k/2)]` as its new name. Note that you have to type in the complete name, it is not possible to exchange only the expressions for the parameters. If the typed name is correct press the return key and now the selected instance has got the new name. The system returns to the cell selection mode in order to let you change the label of another macro. This mode can be aborted by pressing the right mouse button within the workarea.

Entering Wires

↝ 12.4.1

- Wires -
Enter
Delete

Now we turn to the wiring of `SEL[k]` which is very simple. As you can see in the schematic for `CSA[n]` (c.f. figure 4.26) the macro `SEL[n/2+1]` receives at its eastern border the wires representing the carry bits of the sum and the sum plus one of the lower halfs of the operands. This wire has to be connected to all the multiplexers in the row. Therefore we first connect the eastern border of the right instance `SEL[lower(k/2)]` with the eastern border of the schematic. Then we connect the eastern border of the left instance `SEL[upper(k/2)]` with the western border of the right instance. Finally the western border of the left instance is connected with the western border of the schematic. To enter these wires select the entry `Enter` from the `-Wires-` submenu and move the pointer into the workarea. Fix the start point in the middle of the eastern border of the `SEL[lower(k/2)]` with the left mouse button and move the pointer horizontally onto the eastern border of the schematic. If you fix the end point of the wire the input window pops up on the screen. Type in `@carry` to denote that this wire represents the carry values. Now select a new start point for the next wire on the eastern border of the left instance and connect it with the western border of the right instance. At the text input window type in `@carry` to express that this wire represents the continuation of the carry wires which are fed through the `SEL[lower(k/2)]`. In the same way this wire is routed through the left instance, i.e. connect the western border of `SEL[upper(k/2)]` with the western border of the schematic and call it `@carry` as well. Now your schematic should look similar to the one shown in figure 4.31.



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo

CELLDIR: ../Data/Cells/Basic

PARLIB: ...ns/Demo/parcells.lst

SCHEMATIC: SEL[k]

SLICEDIR: ...rch/CadicData/Slices

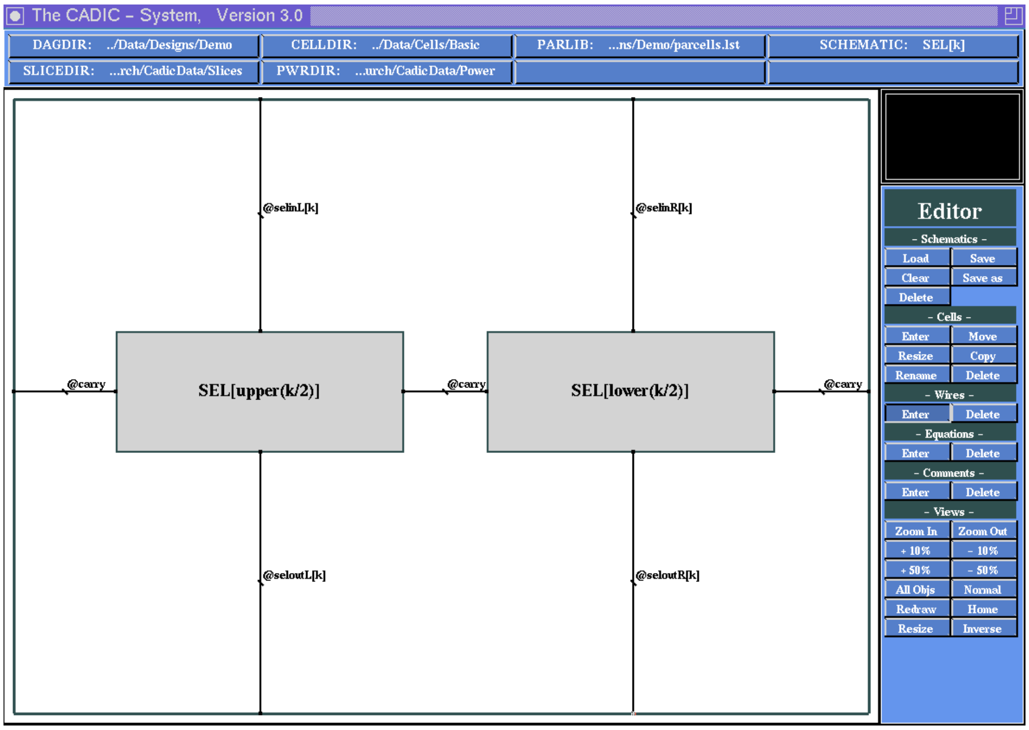
PWRDIR: ...urch/CadicData/Power

@carry
SEL[upper(k/2)]
@carry
SEL[lower(k/2)]
@carry
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.31: Feed through for the carry wires

Now we have drawn the carry wires which control the outputs of the multiplexers. To complete the wiring of `SEL[k]` we draw the data input and output wires. As you can see in the schematic for `CSA[n]` (c.f. figure 4.26) the inputs (outputs) of the selection subcircuit `SEL[n/2+1]` are connected to its northern (southern) border. These wires represent the sum and the sum plus one of the upper half of the operands in alternating sequence. First we draw a wire from the middle of the northern border of the left instance to the northern border of the schematic. Type in `@selinL[k]` for the name of this wire. We choose this name to denote that it represents the left (L) part of the input wires (in) to the selection subcircuit (sel). The number of single binary values represented by this wire depends on the value of the parameter k. This is the reason why we also use this parameter in the name of the wire variable.

Next we connect the southern border of `SEL[upper(k/2)]` with the southern border of the schematic by a wire named `@seloutL[k]`. In the same way


The CADIC – System, Version 3.0
@selinL[k]
@selinR[k]
@carry
SEL[upper(k/2)]
@carry
SEL[lower(k/2)]
@carry
@seloutL[k]
@seloutR[k]
Editor
– Schematics –
Load
Save
Clear
Save as
Delete
– Cells –
Enter
Move
Resize
Copy
Rename
Delete
– Wires –
Enter
Delete
– Equations –
Enter
Delete
– Comments –
Enter
Delete
– Views –
Zoom In
Zoom Out
+ 10%
– 10%
+ 50%
– 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: SEL[k]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

Figure 4.32: Final wiring for the selection subcircuit

we connect the right instance `SEL[lower(k/2)]` with the northern and the southern border of the schematic. These two wires are denoted by the names `@selinR[k]` (northern border) and `@seloutR[k]` (southern border) to indicate that they represent the inputs and outputs of the right part of the selection subcircuit. After you have drawn the last wire you can abort the enter wire mode by pressing the right mouse button twice within the workarea.

Entering Comments

⤳ 12.5.1

– Comments –
Enter
Delete

Now we will place comments to the input and output signals of `SEL[k]`. To do this select the entry `Enter` from the `-Comments-` submenu. Our first comment concerns the input wires at the northern border of the schematic. At the following popup input window type the text

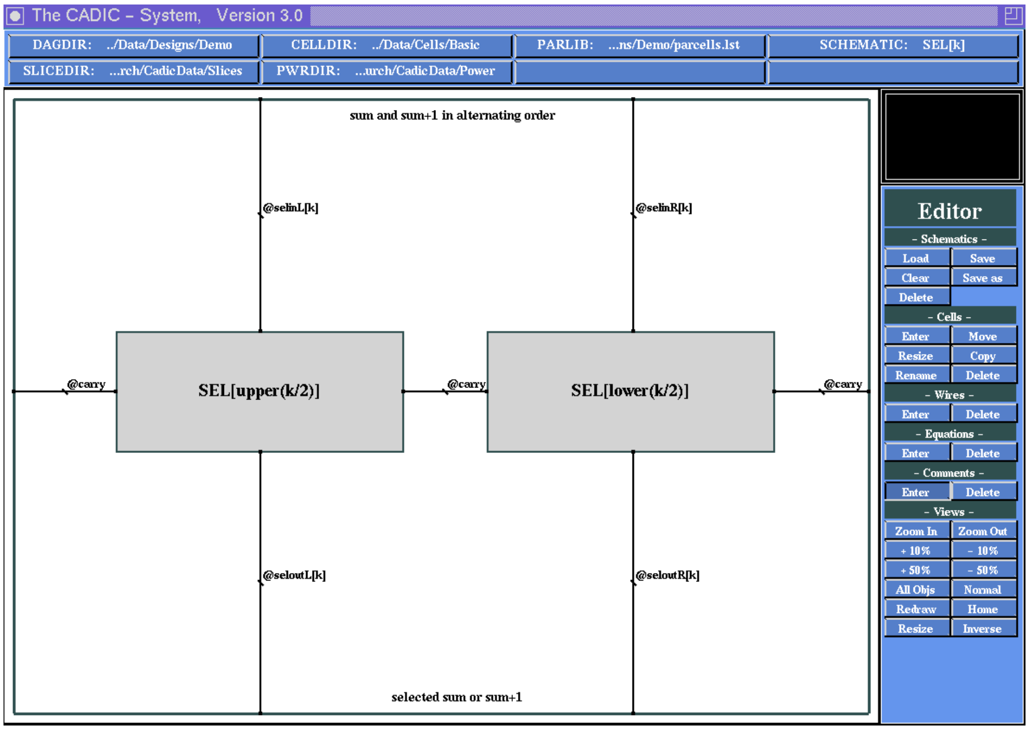
```
sum and sum+1 in alternating order
```

and press the return key. After that you can move the comment text within the workarea. Place it between the two input wires near to the northern border of the schematic and confirm the position with the left mouse button.

Next we place a comment at the output wires at the southern border of the schematic. The popup window appears automatically after you placed the first comment. At its prompt type in

```
selected sum or sum+1
```

and press the return key. The text is now movable inside the workarea. Place it between the two output wires by pressing the left mouse button.


The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: SEL[k]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

sum and sum+1 in alternating order
@selinL[k]
@selinR[k]
@carry
SEL[upper(k/2)]
@carry
SEL[lower(k/2)]
@carry
@seloutL[k]
@seloutR[k]
selected sum or sum+1
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

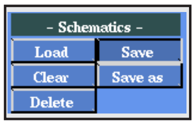
Figure 4.33: Comments for the input and output signals of `SEL[k]`

We will place a last comment at the inputs at the eastern border of the schematic. These control signals represent the carry values of the sum plus one and the sum of the addition of the lower operand parts. That is why we enter the comment `c1Lc0L` for this wire. `c1(0)L` stands for carry (`c`) of the sum plus one (sum) (`1 (0)`) of the addition of the lower (`L`) parts of the operands. After pressing the return key move this comment near to the corresponding wire at the eastern border of the schematic where you can place it with the left mouse button.

Finally you can abort the enter comment mode by simply pressing the return key in the empty input window. You return to the menu selection mode.

Saving the Schematic

⤳ 12.2.2

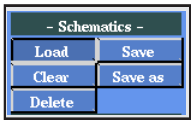


If you have drawn a schematic similar to that in figure 4.33 you can save it by selecting the entry `Save` from the `-Schematics-` submenu. CADIC will put the file `SEL[k].dag` into `DAGDIR`.

4.8.5 Basic Equation for the Selection Subcircuit

Opening a New Schematic

⤳ 12.2.1



To complete the specification of the n bit conditional sum adder we have to draw a schematic for the basic equation of the selection subcircuit. Open a new schematic `SEL[1]` by choosing the `**** New ****` entry in the schematic list which will be displayed if you select the entry `Load` from the `-Schematics-` submenu.

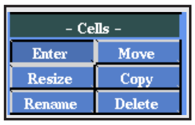
Entering and Manipulating Basic Cells

The basic element of the selection subcircuit has two input bits s_1, s_0 from the sum plus one and from the sum of the corresponding part of the operands. Between these two bits a selection has to be made according to the values of the carry bits c_1 and c_0. Both carries come from the sums of the lower part of the operands. c_1 is the carry of the sum plus one, c_0 is the carry of the sum. The selected values are the outputs a_1 and a_0 of the elementary selection macro. It holds:

$$a_1 \leftarrow (c_1 \wedge s_1) \vee (\overline{c_1} \wedge s_0), a_0 \leftarrow (c_0 \wedge s_1) \vee (\overline{c_0} \wedge s_0).$$

To perform this operations we need two multiplexers from the basic cell library. For this purpose select the entry `Enter` from the `-Cells-` submenu

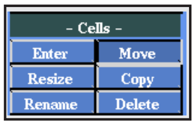
⇝ 12.3.1



. Move the pointer to the entry `MUX` within the basic cell window (upmost) and press the left mouse button when the name is highlighted. Place the cell in the left half of the schematic as shown in figure 4.34. A second multiplexer can directly be selected from the popped up cell selection windows and can be moved to the right half of the schematic. After placing it abort the cell selection by pressing the right mouse button within one of the cell list windows.

The multiplexer has three input pins `I1`, `I2` and `I3` at its northern border and one output pin `O1` at its southern border. `I3` is the selection input with the following functionality:

$$\texttt{O1} \longleftarrow \begin{cases} \texttt{I1} & \text{if } \texttt{I3} = 0 \\ \texttt{I2} & \text{if } \texttt{I3} = 1 \end{cases}$$



⇝ 12.3.2

Before we draw the wiring of `SEL[1]` we change the orientation of the left multiplexer in order to minimize the number of wire crossings. Another reason for this action is to demonstrate how you can change the orientation of an instance. Select the entry `Move` from the `-Cells-` submenu . Although this entry is called `Move` it contains the movement and rotating and flipping of instances. Move the pointer near to the left multiplexer and press the left mouse button when it is highlighted. After that you are in the cell placement mode, i.e. you can select a new position for the cell. But at the same time you can change its orientation by pressing the middle mouse button several time. Each time you press it the instance changes to one of eight possible orientations. The current orientation is indicated by a suffix to the cellname. The cell is in normal orientation if there is no suffix to its name. Pressing the middle mouse button for the first time the cell is turned clockwise by 90 degree, indicated by the suffix `(90->)`.

button presses	suffix	meaning
0		normal orientation
1	`(90->)`	turned clockwise 90^o
2	`(180)`	turned 180^o
3	`(<-90)`	turned counterclockwise 90^o
4	`(^\v)`	flipped at the horizontal axis
5	`(<-90^\v)`	turned counterclockwise 90^o, then flipped at the horizontal axis
6	`(<\>)`	flipped at the vertical axis
7	`(<-90<\>)`	turned counterclockwise 90^o, then flipped at the vertical axis
$8 \equiv 0$		normal orientation

In our case we want to flip the multiplexer at its vertical axis in order to invert the sequence of its input pins. According to the tabular given above you get this orientation by pressing the middle mouse button six times. During this operation you are in the cell placement mode and the instance will follow any movement of the pointer within the workarea. If you have chosen the correct orientation you can move the instance to its previous position and place it by pressing the left mouse button. If you made a mistake in the selection of the orientation you can abort the operation by pressing the right mouse button. Then the cell is returned to its original position and orientation and you will be in cell selection mode again to restart with the operation.

After placing the instance you are in the cell selection mode again in order to change the orientation of another cell. You can abort this mode by pressing the right mouse button within the workarea.

Entering Wires

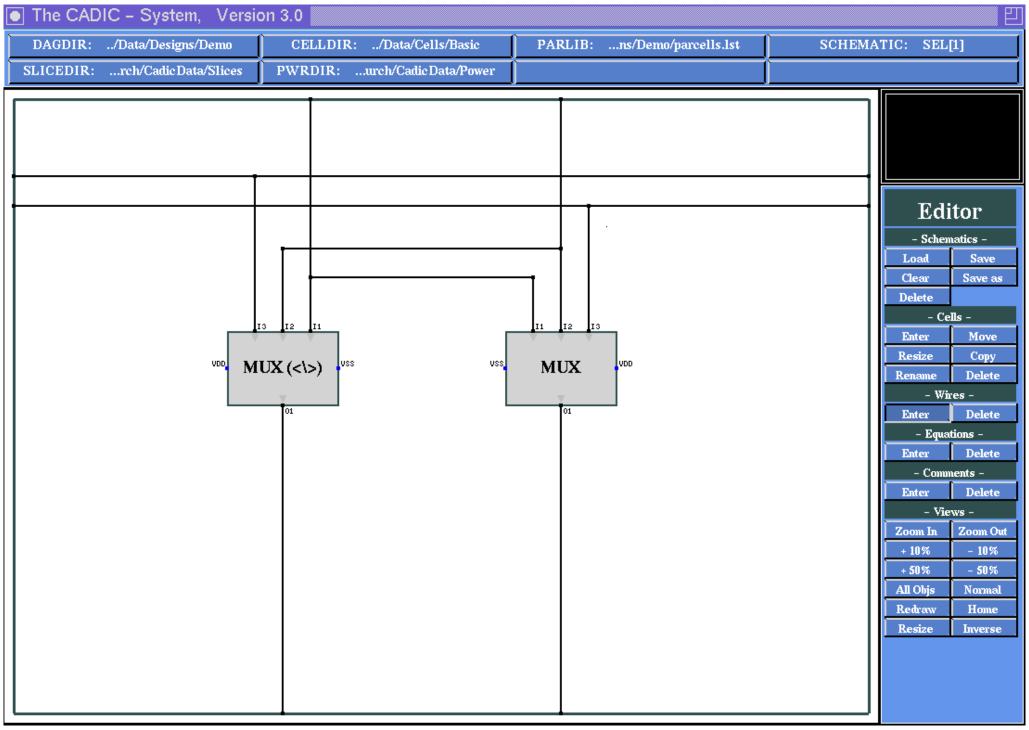
⇝ 12.4.1

- Wires -
Enter
Delete

Now we begin with the wiring of `SEL[1]`. Select the entry `Enter` from the `-Wires-` submenu. First we connect the output pins of the multiplexers with the southern border of the schematic. Move the pointer near to the pin at the southern border of the left multiplexer and press the left mouse button when the crosshair cursor snaps to the pin. Now move vertically down towards the schematic border drawing the rubberbanding line and fix the end point of the wire. Note that you do not have to enter the width of the wire. Every wire which is connected to a pin of a basic cell is automatically given the width 1. In the same way we connect the output pin of the right multiplexer with the southern border of the schematic.

It has been shown above that the value s_1 has to be assigned to the outputs if c_1 or c_0 have the value 1. Respectively the value s_0 is the output if c_1 or c_0 are 0. From the description of the multiplexer it follows that we have to connect both inputs `I1` with the schematic input s_1 and both inputs `I2` with s_0. The selection input `I3` of the left multiplexer has to be connected with c_1 and the selection input of the right multiplexer with c_0. Fix the start point of the next wire at the input pin `I1` of the left multiplexer. Move vertically to the northern border of the schematic and fix the end point of this wire. Now select the input pin `I1` of the right multiplexer move a little bit upwards and then horizontally to the just drawn wire and fix the end point on it, creating an east–branch. To draw the next wire you first have to press the right mouse button once, because we do not want to use the end point of this wire as the new start point. After that we select the input pin `I2` of the right multiplexer and draw a vertical connection with the northern schematic border. From the input pin `I2` of the left multiplexer we draw a connection to the wire starting at the input pin `I2` of the right multiplexer and create a west–branch on this wire. After fixing the end point you must press the right mouse button once in order to return to the selection of a new start point.

Finally we have to enter the wires for the control signals which have to be connected with the `I3` input pins of the multiplexers. It has been shown above that these wires have to be fed through the whole selection subcircuit because they are used to control all the multiplexers in the row. For this purpose we first draw the wires from the eastern to the western border of the schematic. Fix the start point of the first wire (c_1) on the eastern border. Leave enough space to enter the second wire between this one and



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: SEL[1]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

MUX (<\>)
MUX
I3
I2
I1
I1
I2
I3
VDD
VSS
VSS
VDD
O1
O1
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.34: Final wiring of `SEL[1]`

the horizontal parts of the two input signals. Move the pointer horizontally to the western border and fix the end point. At the following input window type in `1` as the width of this new wire.

Now you draw a second connection from the eastern to the western border for the carry signal c_0 below the first one (type in `1` to the input window as the width of this wire).

The connection to the inputs `I3` of the multiplexers is done by first selecting the pin and then moving vertically to the appropriate wire (the left (right) multiplexer has to be connected with the upper (lower) wire) creating a south–branch. Compare your schematic with that shown in figure 4.34.

Entering Comments

⤳ 12.5.1

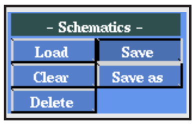
- Comments -
Enter
Delete

Now we place comments at the input and output signals of `SEL[1]` according to the equations shown above. First we label the left input wire at the northern border of the schematic. To do this select the entry `Enter` from the `-Comments-` submenu and type `s1` at the following input window. After you have hit the return key, you can move the comment text within the workarea. Place it near to the connection of the left input wire with the northern border of the schematic. The text `s1` denotes that this signal represents a single bit of the sum plus one. In the same way the right input wire at the northern border is labelled `s0` to show that it is a single bit of the sum.

The schematic `SEL[1]` has two more input wires which represent the incoming carry values. These carries are input from the eastern border and are routed through `SEL[1]` to its western border in order to be available in the selection subcircuit on the left side of the current. The upper one of the two carry wires represents the value of the carry of the sum plus one. Therefore we place a comment `c1L` near to the connection of this wire with the eastern border of the schematic. This comment marks this wire as being the carry (`c`) of the sum plus one (`1`) of the lower part (`L`) of the operands. For analog reasons the lower carry wire is labelled with `c0L`.

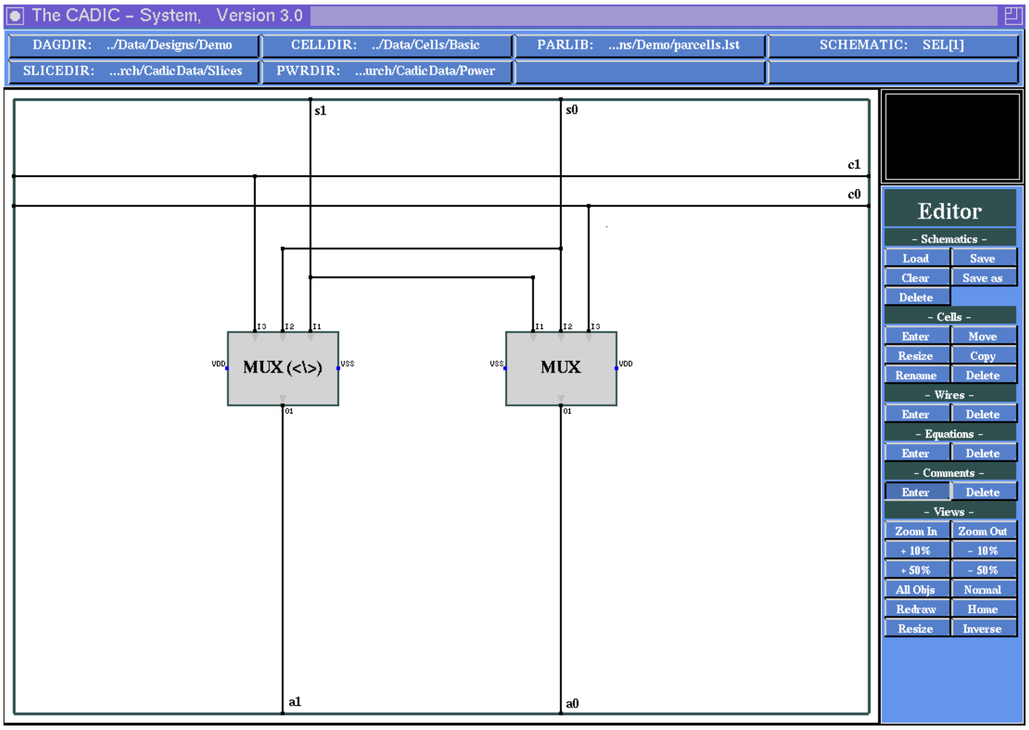
Finally we mark the output wires of `SEL[1]`. According to our equation from above we place the comment text `a1` near to the connection of the left output wire with the southern border of the schematic. The right output wire gets the comment `a0`. Before we save the schematic you should compare your input with that shown in figure 4.35.

Saving the Schematic



To complete the specification of the n bit conditional sum adder we save the last schematic to the file `SEL[1].dag` in `DAGDIR` now. Do this by selecting the entry `Save` from the `-Schematics-` submenu.

Now we have drawn the four schematics needed to describe an arbitrary conditional sum adder for operands of length $n = 2^k$ where $k \geq 0$. Note that you do not have to change any of the inputs to extract e.g. a 32 bit adder instead of a 16 bit adder. This is true if we only consider the functional behaviour of the circuit. In reality the change of the bitlength of the operands influences the timing behaviour of the signals because of



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: SEL[1]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

s1
s0
c1
c0
I3
I2
I1
VDD
MUX (<\>)
VSS
O1
MUX
a1
a0
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.35: Remarks for the input and output signals of `SEL[1]`

increasing fanout for example. We will consider such problems when we describe the integrated tools.

Another problem is the succession of the input and output signals of our design of the conditional sum adder. As shown above the inputs are given in the sequence a_{n-1},b_{n-1},. . ., a_i,b_i,. . .,a_0,b_0 because of the recursive splitting of the operands. This is a restriction if we want to visualize simulation results in order to check the correctness of our design. We will demonstrate those simulations in chapter 6. A more natural sequence for the inputs of the adder would be a_{n-1},. . .,a_0 followed by b_{n-1}, . . .,b_0. In the same way the sequence of the output signals c_1,c_0,$s_{1,n-1}$,$s_{0,n-1}$, . . .,$s_{1,0}$,$s_{0,0}$ which contains the bits of the sum plus one and the sum in alternating order should be changed to the bits of the sum only preceeded by the corresponding carry bit.

We can achieve these signal sequences for the inputs and outputs of the conditional sum adder by two simple subcircuits which only contain wiring elements. These subcircuits which are also parameterized by the length n of the operands can be described by very simple recursive equations as we will show in the following paragraphs.

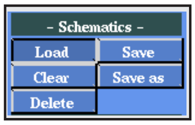
4.8.6 General Equation for Shuffle Subcircuit

We want to specify a circuit with the following functionality: given the input sequence $a_{n-1},\ldots,a_0,b_{n-1},\ldots,b_0$ it should produce the output sequence $a_{n-1},b_{n-1},\ldots,a_0,b_0$. This permutation operation is used very often and is called shuffling of the input wires.

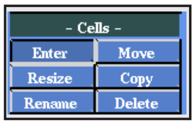
A recursive specification can be derived by the following considerations. To shuffle two sequences a and b of n bits we take the first half of a and the first half of b and shuffle them in an appropriate shuffle subcircuit. In the same way we take the second half of a and the second half of b and shuffle them together.

Opening a New Schematic

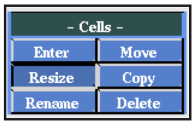
↝ 12.2.1 Open a new schematic for the general recursive equation of the shuffling subcircuit. We call this schematic `SHUFFLE[n]` to show that it can be used to shuffle two input sequences of length n.



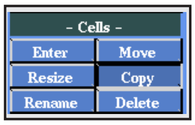
Entering and Manipulating Parameterized Macros



We obtain a simple recursion, because the length of the input sequences of the shuffling subcircuit have the same size as the length of the operands of the conditional sum adder ($n = 2^k$). As shown above we need two parameterized macro cells `SHUFFLE[n/2]`. We will enter the first one by selecting ↝ 12.3.1 the entry `Enter` from the `-Cells-` submenu . Move the pointer into the window with the parameterized macro names (third cell selection window) and select the entry `--- New Cell ---` by pressing the left mouse button when it is highlighted. In the following input window type in the name `SHUFFLE[n/2]` and press the return key. Now you can move the graphical representation of the new parameterized macro cell within the workarea. Place it near to the western border of the schematic. We will create the second instance of `SHUFFLE[n/2]` with the help of the copy operation and you can abort the cell selection mode by pressing the right mouse button within one of the three cell selection windows.



↝ 12.3.3 Before we create a copy we will change the default size of the macro. Select the entry `Resize` from the `-Cells-` submenu. If you move the pointer into the workarea you are in cell selection mode and the only instance within the schematic is highlighted in blue. Select it for the resize operation by pressing the left mouse button. After the selection you can draw a rubberbanding frame which indicates the new size of the macro. Choose an appropriate size and confirm it with the left mouse button. After that you return to cell selection mode which can be aborted by pressing the right mouse button within the workarea.



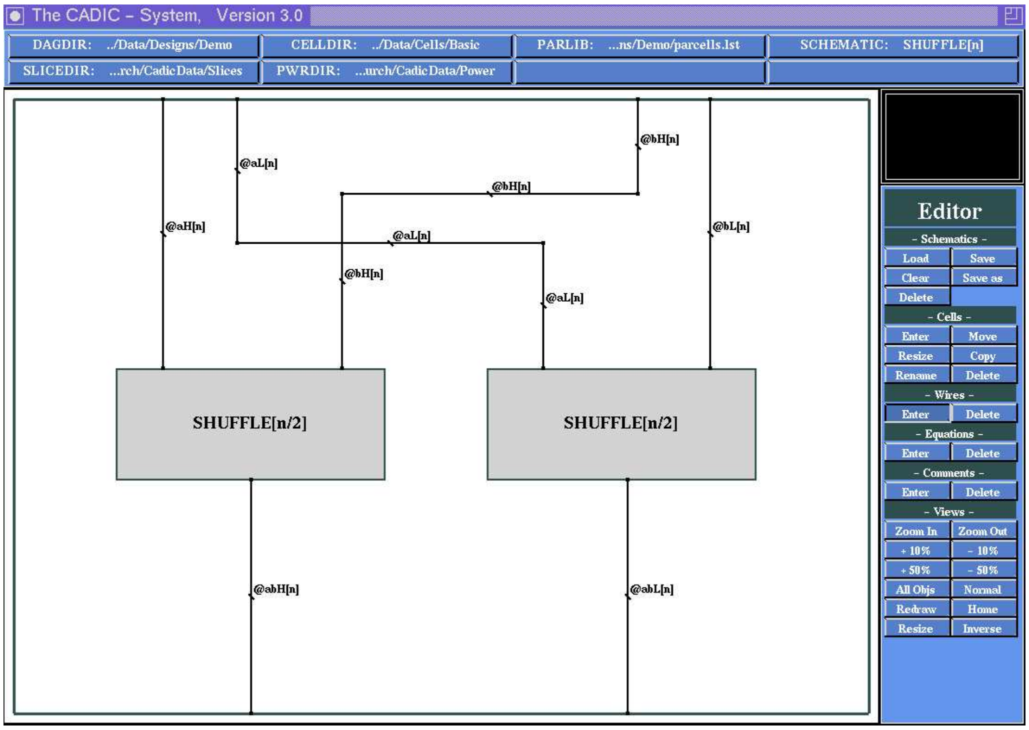
↝ 12.3.4 The second instance is now created by copying the first one. To do this select the entry `Copy` from the `-Cells-` submenu. Move the pointer into the workarea to select the first `SHUFFLE[n/2]` with the left mouse button. Now you get a second macro with the same size and name as the first one. You are in the cell placement mode, so you can choose an appropriate position for the new instance (c.f. figure 4.36) and confirm this with the left mouse button. The following cell selection mode can be aborted by pressing the right mouse button within the workarea.

Entering Wires

- Wires -
Enter
Delete

↝ 12.4.1 We start entering the wiring for `SHUFFLE[n]` at the southern borders of the two instances. These are the output wires of the shuffling subcircuit. To activate enter wire mode select the entry `Enter` from the `-Wires-` submenu. First we select the start point on the southern border of the left `SHUFFLE[n/2]`. Fix it with the left mouse button and move the pointer vertically downwards until it snaps to the southern border of the schematic. After fixing the end point the input window will popup in order to specify the width of this wire. Type in `@abH[n]` to denote that this wire represents the higher part of the shuffled sequence of two input sequences a and b. In the same way you should connect the southern border of the right

`SHUFFLE[n/2]` with the southern border of the schematic. The width of this wire is described by `@abL[n]`.



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: SHUFFLE[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

@bH[n]
@aL[n]
@bH[n]
@aH[n]
@aL[n]
@bL[n]
@bH[n]
@aL[n]
SHUFFLE[n/2]
SHUFFLE[n/2]
@abH[n]
@abL[n]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

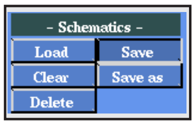
Figure 4.36: Schematic for the general equation `SHUFFLE[n]` of the shuffling subcircuit

Now we turn to the input wires of the two shuffle instances. As shown above we have to shuffle the first half of the first input sequence with the first half of the second input sequence in the left `SHUFFLE[n/2]`. For these purposes we connect the northern border of the left instance with the northern border of the schematic with a wire of width `@aH[n]` to denote that it represents the upper (H) half of the input sequence a. In the same way we connect the northern border of the right instance with the northern border of the schematic by a wire of width `@bL[n]` (c.f. figure 4.36). Next we draw a wire from the right part of the northern border of the left instance near to the point where the wire `@bL[n]` is connected to the schematic border. This can be done by fixing the start point on the border of the left instance and moving the pointer a short distance upwards and then to the right near to the wire `@bL[n]`. Here we fix a wire point by pressing the left mouse button.

Enter `@bH[n]` as the width of this wiring which represents the upper half of the input sequence b. From this point we connect the wire to the northern border of the schematic now. This can easily be done because a fixed end point serves as start point for the next wire. So we have to click one time to fix the end point of the previous wire and use this point to continue drawing. You do not have to type in the width again because it can be derived from the continued wire. In the same way you should connect the northern border of the right `SHUFFLE[n/2]` with a position on the northern border of the schematic near to the position of the wire `@aH[n]`. The width of this wire is denoted by `@aL[n]`. Abort the enter wire mode by pressing the right mouse button twice within the workarea.

Saving the Schematic

↝ 12.2.2



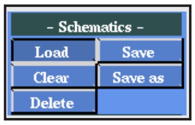
In this schematic we do not enter comments for the reasons of simplicity. Now you can save the general equation for the shuffling subcircuit into `DAGDIR` by selecting the entry `Save` from the `-Schematics-` submenu.

4.8.7 Basic Equation for Shuffling Subcircuit

The basic equation for the shuffling subcircuit concerns the merging of two input sequences of length 2. It is obvious that in case of sequences of length 1 we have nothing to do (the input is equal to the output). For two sequences of length 2 a_1,a_0 and b_1,b_0 the output sequence is a_1,b_1,a_0,b_0. This can be achieved by a simple schematic input which only contains wiring elements.

Opening a New Schematic

↝ 12.2.1



For the basic equation of the shuffling subcircuit we create a new schematic named `SHUFFLE[2]`. This denotes that it shuffles two input sequences of length 2 (compare this with the general equation where `SHUFFLE[n]` shuffles two sequences of length n). Open the new schematic by selecting `**** New ****` from the list of schematic names which will be displayed when you select the entry `Load` from the `-Schematics-` submenu. Type `SHUFFLE[2]` as input to the popup window and press the return key.

Entering Wires

- Wires -
Enter
Delete

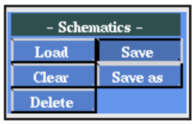
↝ 12.4.1 In this schematic we need no macros nor basic cells, so we proceed with entering the wiring. As shown above `SHUFFLE[2]` has four input and four output signals. The output values are the same as the inputs, but their order has changed. From the comparison between the order of the inputs and the outputs we obtain the desired schematic. First we notice that the first output signal a_1 is the same signal as the first input signal. Therefore we draw a vertical wire from the northern border to the southern border of the schematic. To do this select the entry `Enter` from the `-Wires-` submenu. Move the pointer onto the northern border of the schematic and fix the start point near to the western border by pressing the left mouse button. Draw the rubberbanding line vertically down towards the southern border and fix its end point in the same way. Because this wire represents one single bit of the input sequence we type `1` as the input to the window asking the width of the wire.

For the same reasons we can draw the wire for the last input bit b_0 near to the eastern border straight from the northern to the southern border of the schematic. The width of this wire also is `1`.

As shown above the two bits in the middle of the input sequence a_0 and b_1 have to be exchanged. We achieve this by drawing two crossed wires. Fix the start point for the input a_0 at the northern border of the schematic right beside a_1 and move the pointer vertically down a certain distance. Fix the end point and type in `1` for its width. From this end point we continue the wire horizontally towards the right wire for b_0 and then down to the southern border of the schematic where it is fixed by pressing the left mouse button (c.f. figure 4.37). In the same way you should draw the wire for the third input bit b_1 from the northern to the southern border of the schematic creating a crossing with the previous wire. Now your schematic should look like that shown in figure 4.37.

Saving the Schematic

↝ 12.2.2 In this schematic we do not enter comments for the reasons of simplic-



ity. Now you should save the basic equation for the shuffling subcircuit to the file `SHUFFLE[2].dag` in `DAGDIR` by selecting the entry `Save` from the `-Schematics-` submenu.

The shuffling subcircuit generates the correct input sequence a_{n-1},b_{n-1},. . .,a_0,b_0 for our design of the conditional sum adder. This subcircuit helps us to interprete the simulation results in chapter 6 because we can keep the bits of the operands a and b together.

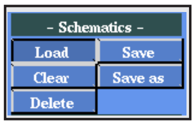
For a similar reason we combine the bits of the output values of the conditional sum adder. As shown above the sequence of the output bits is c_1,c_0, $s_{1,n-1}$,$s_{0,n-1}$,. . .,$s_{1,0}$,$s_{0,0}$. The first two bits represent the carries of the sum plus one and the sum of the addition. The next $2n$ bits represent the sum plus one and the sum in alternating order. As result we only want to have the carry and the bits of the sum c_0,$s_{0,n-1}$,. . .,$s_{0,0}$ as outputs of our addition circuit. To perform this task we will now specify a subcircuit which cuts the not desired wires.

4.8.8 General Equation for the Cutting Subcircuit

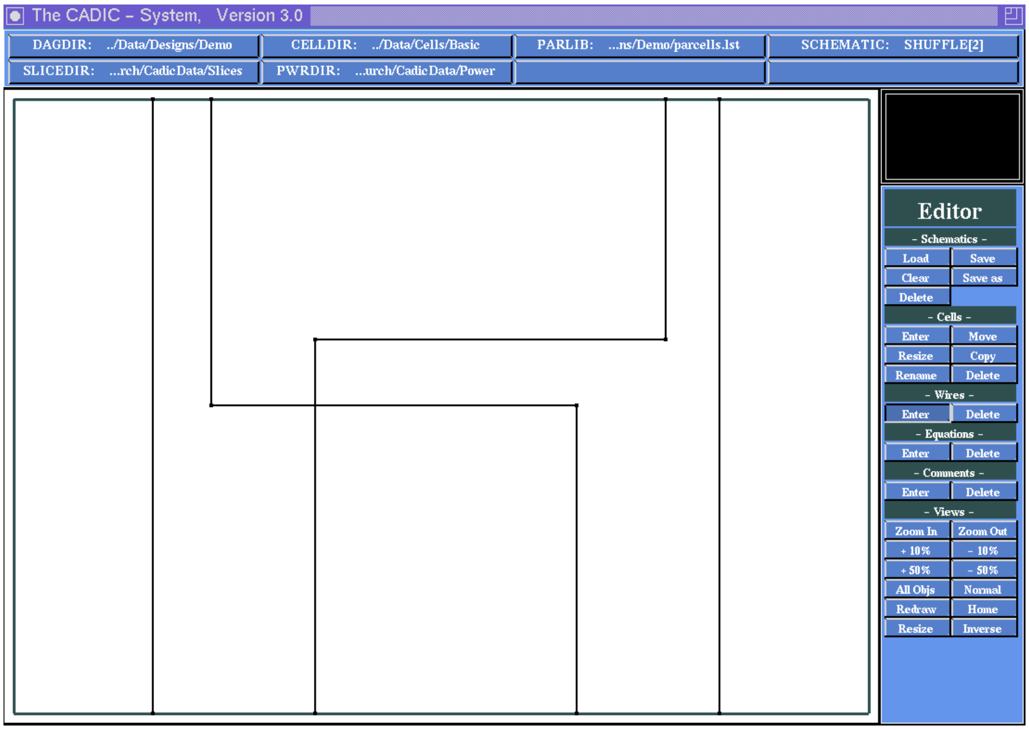
Similar to the row of multiplexers from section 4.8.4 this subcircuit can be recursively specified. The task of the whole circuit is to cut each wire on an odd position from a sequence w_1,w_2,w_3,. . .,w_k in order to generate the sequence w_2,w_4,. . .,w_k (if k is an even number as in our special case). That means we have to create a row of $\frac{k}{2}$ elements where each element cuts the first wire and feeds through the second.

Opening a Schematic

↝ 12.2.1



We use the schematic `SEL[k]` to create the new equation because the structure of the general equation of the cutting subcircuit is very similar to that of the row of multiplexers. To open it select the entry `Load` from the `-Schematics-` submenu and move the pointer onto the name `SEL[k]`. Select it by pressing the left mouse button when it is highlighted.


The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: SHUFFLE[2]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.37: Schematic for the basic equation `SHUFFLE[2]`

Saving a Schematic Under a New Name

↝ 12.2.3

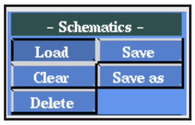
- Schematics -
Load
Save
Clear
Save as
Delete

❗

In the workarea the schematic input of `SEL[k]` is visible. We want to copy this schematic to a new sheet in order to construct the cutting subcircuit. You can do this by selecting the entry `Save As` from the `-Schematics-` submenu. In the following input window you should type in a new name for the schematic, here it should be `CUT[k]`. Note that after you pressed the return key the name of the loaded schematic still is `SEL[k]`.

Opening a Schematic

↝ 12.2.1


- Schematics -
Load
Save
Clear
Save as
Delete

In order not to overwrite `SEL[k]` first we have to load the copy which is saved under the name `CUT[k]`. To do this select the entry `Load` from the `-Schematics-` submenu. In the list window move the pointer onto the name `CUT[k]` and press the left mouse button when the name is highlighted. Note that you will see the same schematic as before but the schematic name in the upper right corner of the graphical surface has changed to `CUT[k]` which is now the schematic under construction.

Deleting Wires

↝ 12.4.2

- Wires -
Enter
Delete

Before we draw all the wires for `CUT[k]` we have to delete the wires from `SEL[k]`. To do this select the entry `Delete` from the `-Wires-` submenu. If you move the pointer into the workarea you are in the wire selection mode. The wire nearest to the pointer is highlighted (it changes its colour from yellow to cyan). By pressing the left mouse button you can delete the highlighted wire. After the deletion operation a new wire is highlighted which is the closest wire to the pointer position. Proceed with this operation until all wires are deleted. After the deletion of the last wire the wire selection mode is automatically terminated and you return to menu selection mode. Alternatively the wire deletion mode can be terminated by pressing the right mouse button within the workarea.

Deleting Comments

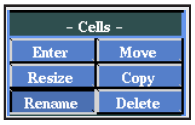
↝ 12.5.2

- Comments -
Enter
Delete

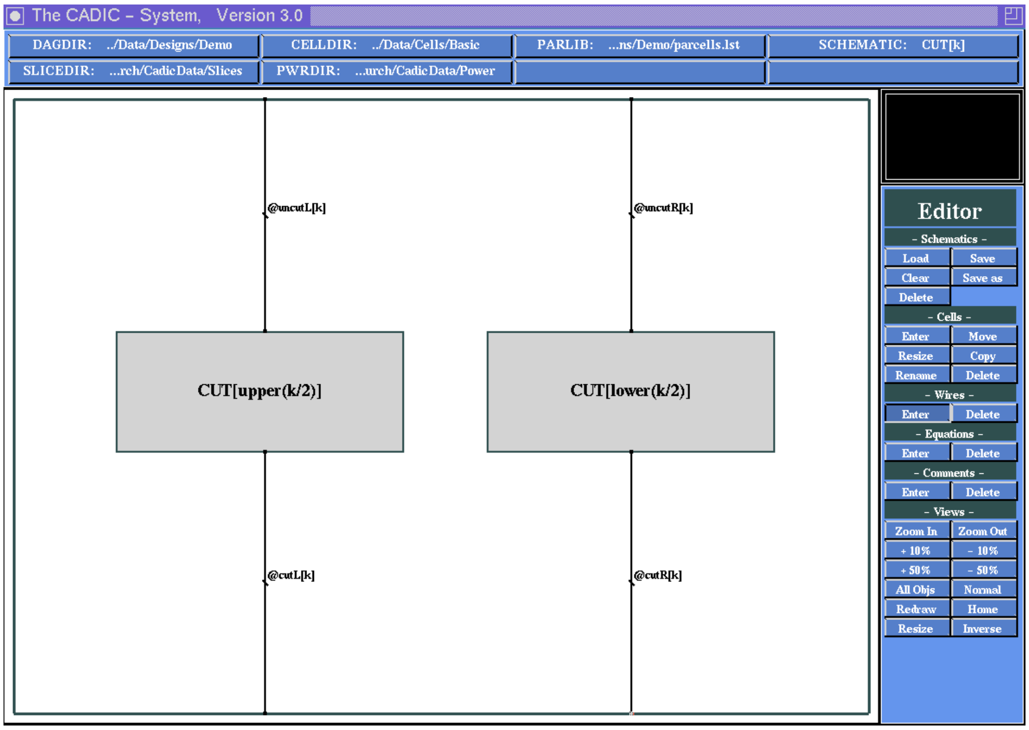
Just as the wires we have to delete the two comment texts in the schematic before we change the labels of the macros. To do this select the entry `Delete` from the `-Comments-` submenu. If you move the pointer into the workarea you are in comment selection mode and the comment text nearest to the pointer is highlighted (its colour changes from white to cyan). By pressing the left mouse button this text can be erased from the schematic. After the deletion the next comment text which is nearest to the pointer is highlighted. Proceed with this operation until all comments are erased. After the deletion of the last comment the function is automatically terminated as well as when you press the right mouse button within the workarea.

Manipulating Parameterized Macros

↝ 12.3.5



Next we have to rename the two instances which are still labelled `SEL[upper(k/2)]` and `SEL[lower(k/2)]`. According to the name `CUT` of our schematic we have to change the names of these instances. Select the entry `Rename` from the `-Cells-` submenu. If you move the pointer into the workarea you are in cell selection mode where the instance closest to the pointer position is highlighted in blue. First we select the left instance and press the left mouse button. In the following input window we type in `CUT[upper(k/2)]` as its new name and press the return key to confirm the input. Now select the right instance in the same way and change its name to `CUT[lower(k/2)]`. After that you can abort cell selection mode by pressing the right mouse button within the workarea.



DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CUT[k]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

The CADIC – System, Version 3.0
@uncutL[k]
@uncutR[k]
CUT[upper(k/2)]
CUT[lower(k/2)]
@cutL[k]
@cutR[k]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.38: Graphical input for the general equation of the cutting subcircuit `CUT[k]`

Entering Wires

⤳ 12.4.1

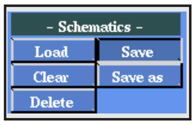
- Wires -
Enter
Delete

The wiring of `CUT[k]` is very simple and only consists of vertical wires connecting the two instances with the schematic borders. To draw it select the entry `Enter` from the `-Wires-` submenu. First we connect the northern border of the left instance with the northern border of the schematic. This wire represents the input signals to `CUT[upper(k/2)]`, i.e. from this bundle each wire at an odd position has to be cut. At the prompt for the width of this bundle type in `@uncutL[k]` to denote that it represents the left half of the uncut wires. Now select the start point for the next wire at the southern border of the left instance and move the pointer vertically down towards the southern border of the schematic where you fix the end point of this wire. Type in `@cutL[k]` as its width in order to show that the output of the left instance are the cut wires.

In the same way we connect the right instance with the corresponding schematic borders. These wires have the width `@uncutR[k]` (`@cutR[k]`) for the connection between the northern (southern) borders. Your graphical input should look similar to that shown in figure 4.38. Abort the wire enter mode by pressing the right mouse button twice within the workarea.

Saving the Schematic

↝ 12.2.2



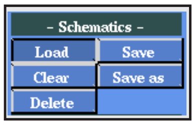
Save the schematic `CUT[k]` into `DAGDIR` by selecting the entry `Save` from the `-Schematics-` submenu.

4.8.9 Basic Equation for the Cutting Subcircuit

The basic element for the cutting subcircuit is very simple. It takes two input signals w_i,w_{i+1} (where i is an odd number) of the whole input sequence, cuts the first one and feeds through the second w_{i+1} as its output signal.

Opening a New Schematic

↝ 12.2.1



For the basic equation of the cutting subcircuit we now open a new schematic `CUT[1]`. Select the entry `Load` from the `-Schematics-` submenu and move the pointer to the position of `**** New ****` within the list window for the schematic names. When it is highlighted press the left mouse button. At the following input window type `CUT[1]` as the new name and press the return key.

Entering Wires

⤳ 12.4.1

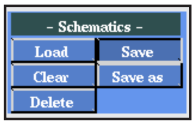
- Wires -
Enter
Delete

Because this schematic does not contain macros we proceed with entering the wiring just as we did it in the case of `SHUFFLE[2]`. `CUT[1]` has two input signals from which the first one is cut. To draw the needed wires select the entry `Enter` from the `-Wires-` submenu. Move the pointer into the workarea near onto the middle of the northern border of the schematic and fix the start point of the first wire with the left mouse button. Move vertically down until you have reached the middle of the schematic where we fix the end point of this wire. Now we have drawn a wire which starts at the northern border of the schematic and ends within the schematic. As you have already learned during the input of the general equation for the conditional sum adder `CSA[n]` the end point of this wire is a legal operation within the underlying calculus. Cutting a wire to an open end is called a projection operation. For the width of this wire type in `1` at the following input window.

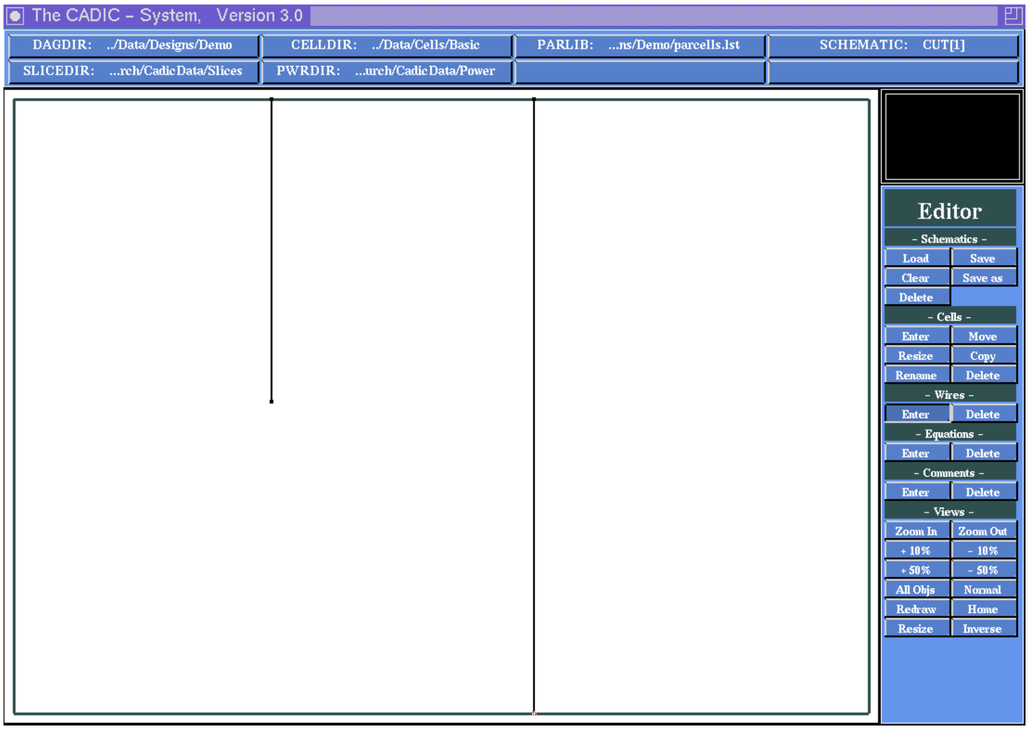
The second wire in `CUT[k]` has to connect the northern and the southern border of the schematic. Before you can fix the start point of this wire you have to press the right mouse button. This is necessary to tell the editor that you want to fix a new start point for the next wire and that you do not want to continue the just drawn wire (as we did it in the case of the wiring of `SHUFFLE[2]`). Note that you have to select a start point of the wire on the northern border right beside the start point of the previous wire. The width of this wire is `1` as well. Abort the enter wire mode by pressing the right mouse button twice within the workarea.

Saving the Schematic

⤳ 12.2.2



If you have drawn a schematic similar to that shown in figure 4.39 you can save it into `DAGDIR` by selecting the entry `Save` from the `-Schematics-` submenu.



The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CUT[1]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

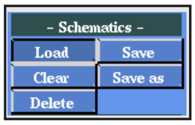
Figure 4.39: Projection of a single wire in the basic equation of the cutting subcircuit `CUT[1]`

4.8.10 Equation for the complete n bit adder

In this section we will combine the main part of the conditional sum adder, i.e. the circuit `CSA[n]` for n bit operands with the two wiring subcircuits `SHUFFLE[n]` and `CUT[k]` to obtain an addition circuit with the following functionality. It receives two binary vectors $a = a_{n-1} \dots a_0$ and $b = b_{n-1} \dots b_0$ as inputs and generates an output vector of the form $s = cs_{n-1} \dots s_0$ with $s = a + b$. c represents the carry bit of the summation. We choose this form of the adder because it allows us to control simulation results very easily. We will demonstrate this in chapter 6.

Opening a New Schematic

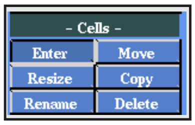
$\rightsquigarrow$ 12.2.1



- Schematics -
Load
Save
Clear
Save as
Delete

For the complete n bit adder we use a simple parameterized equation (not a recursive specification). That is the reason why we only need one equation where we map the parameters of the subcircuits in an appropriate way. We need no basic equation in this case. Open a new schematic `CSADDER[n]` by selecting the entry `Load` from the `-Schematics-` submenu and typing this name at the input window which pops up when you select the entry `**** New ****` in the schematic list window.

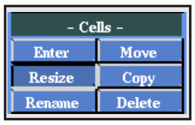
Entering and Manipulating Parameterized Macros



↝ 12.3.1

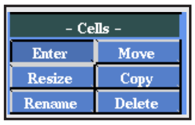
We need three parameterized macro cells in `CSADDER[n]` which will be vertically arranged. We begin with the shuffling of the operands a and b. The length of the operands is n that is why we need an instance `SHUFFLE[n]` to shuffle two sequences of length n. Select the entry `Enter` from the `-Cells-` submenu . Now choose the entry `--- New Cell ---` from the cell list window for the parameterized macros (bottom window). At the input window type the name `SHUFFLE[n]` as input and move the appearing macro to a position near to the northern border of the schematic. Before we select the other macros we want to resize this first one. Therefore you should abort cell selection mode by pressing the right mouse button within one of the three popup windows.

↝ 12.3.3



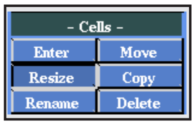
Now select the entry `Resize` from the `-Cells-` submenu. Move the pointer into the workarea and press the left mouse button when the instance `SHUFFLE[n]` is highlighted. Now you can draw the rubberbanding frame to the new macro size. We select a small height for this macro to indicate that it only contains wiring elements and no basic cells. If you have chosen a size similar to that shown in figure 4.40 can should confirm the size with the left mouse button. After that abort the cell selection mode by pressing the right mouse button within the workarea.

↝ 12.3.1



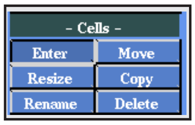
Now we enter the macro for the main part of our adder which is an instance of the general equation for the conditional sum adder. Select the entry `Enter` from the `-Cells-` submenu again in order to create a new parameterized macro (entry `--- New Cell ---` from the third popup window). Type `CSA[n]` at the input window and move the macro with its upper left corner below the instance `SHUFFLE[n]`. Leave enough space between the two instances for the wiring and confirm the position with the left mouse button. After that abort cell selection mode (right mouse button in one of the popup windows) in order to resize `CSA[n]`.

↝ 12.3.3



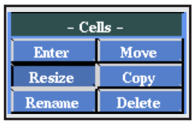
Select the entry `Resize` from the `-Cells-` submenu and move the pointer into the workarea. When the instance `CSA[n]` is highlighted press the left mouse button and select a new size for this macro. Choose the same width as for `SHUFFLE[n]` but make this macro much higher in order to denote that it contains the main part of the whole adder (c.f. figure 4.40). Confirm the new size with the left mouse button and abort any further cell selection with the right mouse button.

↝ 12.3.1



The third macro we need is an instance of the cutting subcircuit. To create it select the entry `Enter` from the `-Cells-` submenu and move the pointer onto the position `--- New Cell ---` in the third popup window. As shown above the output of `CSA[n]` is a sequence of length $2 \cdot (n+1)$ because we have two sums of length n and one carry bit for each of the sums. That is the reason why we need an instance `CUT[n+1]` of the cutting subcircuit (type `CUT[n+1]` at the popup input window). Move the new macro with its upper left corner under the `CSA[n]` and leave enough space for the wiring. After you confirm the position with the left mouse button the cell selection windows popup again. We do not need another instance so you can abort cell selection by pressing the right mouse button within one of the three popup windows.

↝ 12.3.3



The last operation concerning the macros consists in resizing the instance `CUT[n+1]`. Select the entry `Resize` from the `-Cells-` submenu and move the pointer into the workarea so that `CUT[n+1]` is highlighted. Confirm the selection by pressing the left mouse button. With the help of the rubber-banding frame we now select a new size for this macro. Choose the same width as for the other two instances but a much smaller height than that of `CSA[n]`. The height can approximately be the same as that of `SHUFFLE[n]` to denote that this subcircuit only contains wiring elements as well. Compare the sizes of the macros with figure 4.40 and abort cell selection mode if they are correct (right mouse button in workarea).

Entering Wires

⇝ 12.4.1

- Wires -
Enter
Delete

To draw the wiring of `CSADDER[n]` select the entry `Enter` from the `-Wires-` submenu. First we draw the input wires of the instance `SHUFFLE[n]`. Fix a start point at the left half of the northern border of `SHUFFLE[n]` and move the pointer vertically up onto the northern border of the schematic. After fixing the end point with the left mouse button you should type `@a[n]` as width of this wire to denote that it represents the first operand a of length n. In the same way we draw a second wire from the northern border of `SHUFFLE[n]` to the northern border of the schematic. This wire should be located right beside the previous wire. For its width type in `@b[n]` to denote that it represents the second operand b of length n.

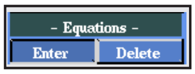
Next we connect the southern border of `SHUFFLE[n]` with the northern border of `CSA[n]`. For the width of this wire type in `@ab[n]` to denote that it represents the bits of the operands a and b in alternating order as it is generated by the shuffling subcircuit.

In the same way we draw a wire from the southern border of `CSA[n]` to the northern border of `CUT[n+1]`. We describe the width of this wire by `@csums[n]`, because the output of `CSA[n]` is the sum and sum plus one, preceeded by the corresponding carries.

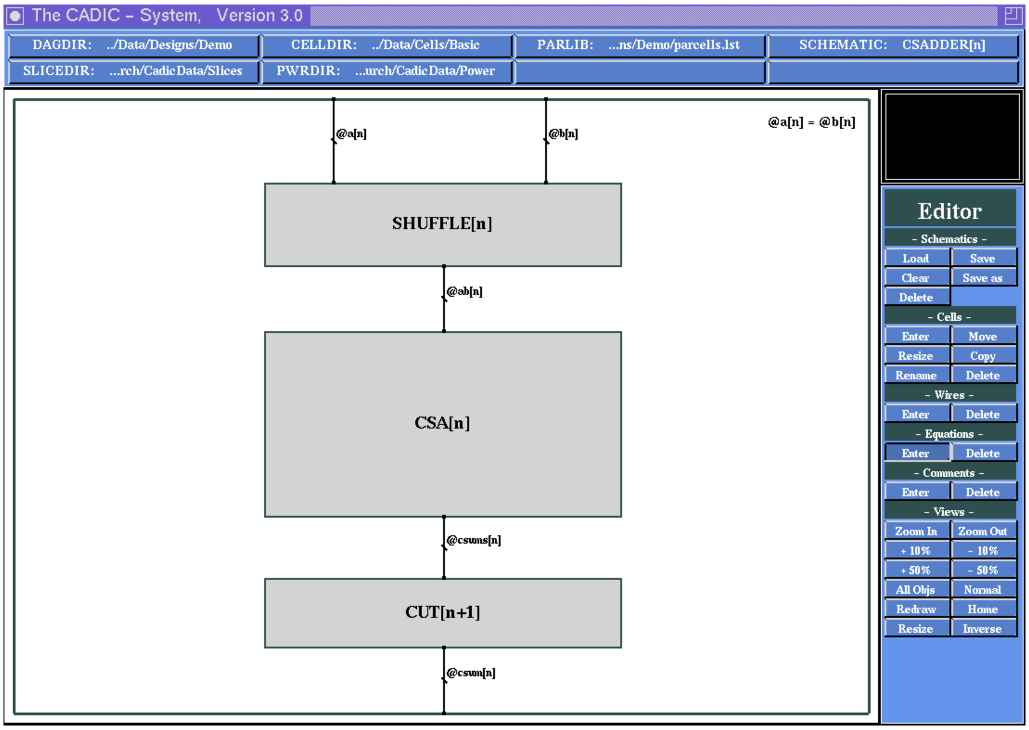
The last wire of this schematic connects the southern border of the instance `CUT[n+1]` with the southern border of the schematic. For the width of this wire type in `@csum[n]` to denote that it only represents the bits of the sum including the leading carry bit. Finally abort the enter wire mode by pressing the right mouse button twice within the workarea.

Entering an Equation

⇝ 12.6.1



In the wiring above we have specified the width of the input operands by the wire variables `@a[n]` and `@b[n]`. The specification of the n bit conditional sum adder will give us the equation that `@a[n] + @b[n] = 2n`. This equation is not sufficient to uniquely determine the value of the two variables. One possible solution would be `@a[n] = 2n` and `@b[n] = 0` but this is not the desired one. To obtain a unique solution we enter an equation about the width of the two variables. To do this select `Enter` from the `-Equations-` submenu. In the same way as you entered a comment text now you can type in an equation about relationships between wire variables in the popup window. Type in `@a[n] = @b[n]` and press the return key. By this equation we tell the system that the width of operand *a* is equal to that of operand *b*. The equation is movable within the workarea like comment text. Select an appropriate place like that shown in figure 4.40 and confirm the position by pressing the left mouse button. After that the input window will popup again. Because we do not need any further equation in our specification just press the return key to the empty input window in order to abort the enter equation mode.


The CADIC – System, Version 3.0

DAGDIR: ../Data/Designs/Demo
CELLDIR: ../Data/Cells/Basic
PARLIB: ...ns/Demo/parcells.lst
SCHEMATIC: CSADDER[n]
SLICEDIR: ...rch/CadicData/Slices
PWRDIR: ...urch/CadicData/Power

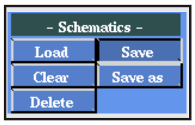
@a[n] = @b[n]

@a[n]
@b[n]
SHUFFLE[n]
@ab[n]
CSA[n]
@csums[n]
CUT[n+1]
@csum[n]
Editor
- Schematics -
Load
Save
Clear
Save as
Delete
- Cells -
Enter
Move
Resize
Copy
Rename
Delete
- Wires -
Enter
Delete
- Equations -
Enter
Delete
- Comments -
Enter
Delete
- Views -
Zoom In
Zoom Out
+ 10%
- 10%
+ 50%
- 50%
All Objs
Normal
Redraw
Home
Resize
Inverse

Figure 4.40: Graphical input for `CSADDER[n]` with an additional equation for unique specification of the wire variables `@a[n]` and `@b[n]`

Saving the Schematic

⤳ 12.2.2 You can check all the operations we used above (entering and resizing macros



and entering the wires) by comparing your schematic with that shown in figure 4.40. If everything is correct you can save the schematic into `DAGDIR` by selecting the entry `Save` from the `-Schematics-` submenu.

4.8.11 Summary

Now we have completed the specification of the n bit conditional sum adder and we will close our editing session. This can be done by simply pressing the right mouse button within the menuline. Then the submenu for the graphical editor will disappear and the system's main menu is displayed again.

Note that our design of the conditional sum adder only works for parameter values $n = 2^k$. If you enter any other value for n, the system may not be able to construct the corresponding adder. For more information about selecting parameter values you can read the section 5.8.

⤳ 5.8

Here you can stop your tour through the CADIC system by selecting the entry `Exit` from the main menu. You can also proceed with using another tool of the system in order to analyze or synthesize your design. From this point of the manual you can skip to another chapter describing the special tool. We recommend to you to continue with chapter 5 where the extraction of a hierarchical representation for fixed values of the design parameters is shown. This hierarchical representation is the basic structure for the integrated tools. It is used to generate and visualize the results during the design process.