6

Logic Simulation

6.1 Introduction

The simulation tool we describe in this chapter allows the user to check the functional behaviour of a design. Only the logical function of the basic cells is taken into account, there are no timing aspects, e.g. delays of the gates and the wires, represented in the simulation model. This simulation tool gives you a basic method to check the correctness of your design.

More detailed information about the timing behaviour of your design will be given by a simulation tool which will be included in the next distribution of the CADIC system. At this moment you can use one of the conversion routines to create a standard exchange format (e.g. EDIF 2 0 0 or GHDL), as they are described in chapter 11 and perform the simulation with a commercial design system.

6.2 Getting Started

If you use this simulation tool you have to consider some preconditions on your design. First of all you may only simulate combinational circuits. The design must not have virtual cycles (c.f. figure 6.1) and the data flow on a bundle of wires must be unique for all single wires of this bundle (c.f. figure 6.2).

If the first condition is violated the system is not able to calculate a topol-

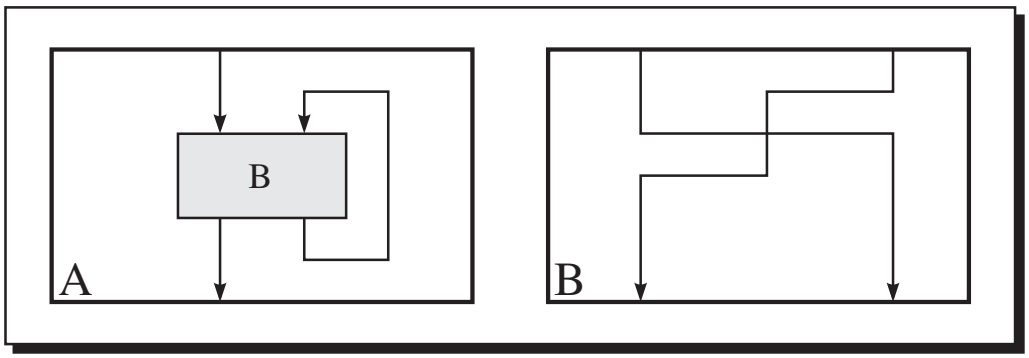


Figure 6.1: Macro A contains virtual cycle

gical sorting of the subcircuits for at least one hierarchy level. Because the simulation is done by a C program this sorting is needed to create the correct order of the function calls for the subcircuits as you will see in the following.

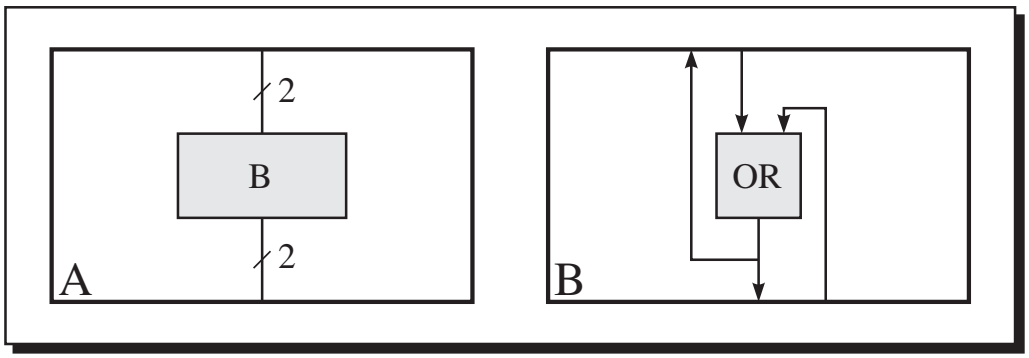


Figure 6.2: Two busses in macro A with non unique data flow

The second condition comes from the objects of the underlying netlist data structure which does not allow single subnets of a bus to have different directions. Both restrictions will be erased within a new simulation tool which is integrated in the new release of CADIC . But nevertheless this simulation provides a comprehensive method for a first checking of a design.

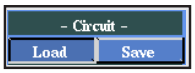
6.3 Sample Session

In this sample simulation session we will first analyze the fulladder design to show the basic functions of the simulation tool. Afterwards we will check the design of the n bit conditional sum adder from chapter 4. The simulation of a design usually takes the following steps

- load and prepare the design for simulation
- start the simulation for a single pattern or a list of patterns
- examine the simulation results by tracing through the circuit hierarchy

In the following we will explain how to perform these steps with the help of our special design example, where not all functions of the simulation tool are needed.

6.3.1 Load and Prepare a Design for Simulation



First we have to load and prepare a circuit for simulation. To do this select the entry `Load` from the submenu `-Circuit-`. After pressing the left mouse button you will see the list of schematics in your `DAGDIR` directory. From this list you should select the entry `FullAdder` by pressing the left mouse button onto the highlighted text.

Again you can read the messages during the loading of `FullAdder` and the construction of the hierarchical data structure. Finally you can read the message

```
Circuit "FullAdder" loaded, Total Data Size:  14698 Bytes
```

Now `FullAdder` is loaded and it will be prepared for simulation. The system will calculate the data flow through the circuit and will generate a C module for the execution of the circuit operations as it is shown in figure 6.3.

This module will be created in the directory given by the environment variable `SIMDIR`. In the message area you can see the following information:

```
cannot open file ../Data/Sim/FullAdder.out
```

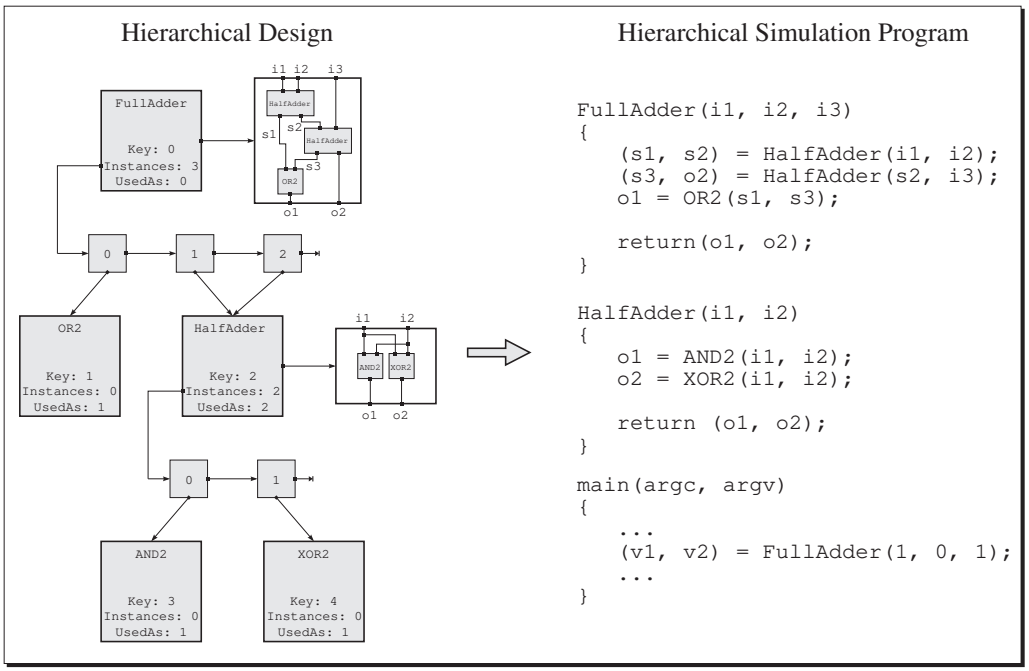


Figure 6.3: Creation of a hierarchical simulation program

This means that the executable simulation program does not yet exist. Here `../Data/Sim` is the value of the environment variable `SIMDIR` and `FullAdder.out` is the name of the executable simulation program. This program is created by translating the C module `../Data/Sim/FullAdder.c` and linking subsequently with an object file for the basic cells which is located in the `CELLDIR` directory and is called `basiccells.o`. For the compilation and the linking of the simulation program CADIC uses the same compiler as during the installation of the system . If you get a copy of the executable system, then GNU `gcc` is the default compiler.

~> 2

During the creation of the executable simulation program the system will display the message

`Generation of the simulation program:`

In the workarea the circuit is not fully shown, until the process of translating and linking of the simulation program has terminated. If the system displays the message

```
--- Finished
```

the executable file is created and the system is ready for starting the circuit simulation. At this moment your workarea should look like that shown in figure 6.4.

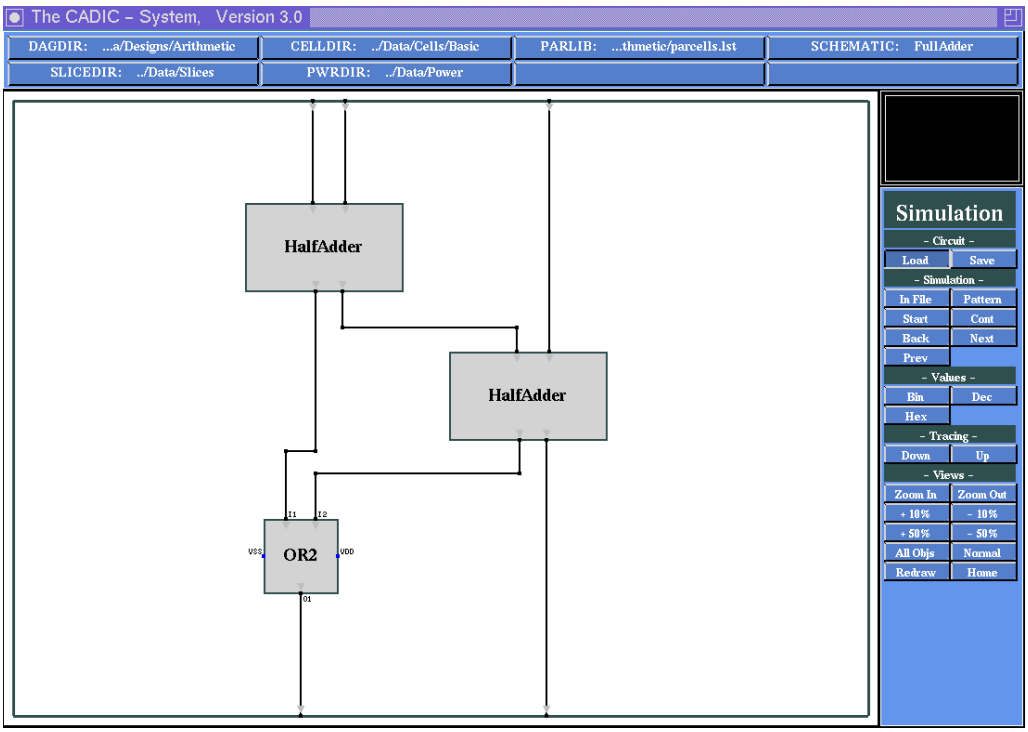


Figure 6.4: Circuit `FullAdder` loaded and prepared for simulation

In figure 6.4 you will see that the system has calculated the data flow through the circuit. This is indicated by the small arrows at the inputs and outputs of the two instances of `HalfAdder`.

6.3.2 Simulation of a Single Pattern

First we want to simulate the circuit for a single input pattern. For this purpose select the entry `Pattern` from the `-Simulation-` submenu. After activating this function the system asks you for the values at the input pads of the circuit. Your screen should look like that shown in figure 6.5. First you have to enter the input value for the most right pad at the northern

border of `FullAdder` which is highlighted by a small square. At the prompt window you can give a binary input value for this pad. Just type in

```
Give Input Value for this Pad <1        >
```

as it is shown in figure 6.5. Note that the binary string must have a length of 1, because of the width of the corresponding input wire.

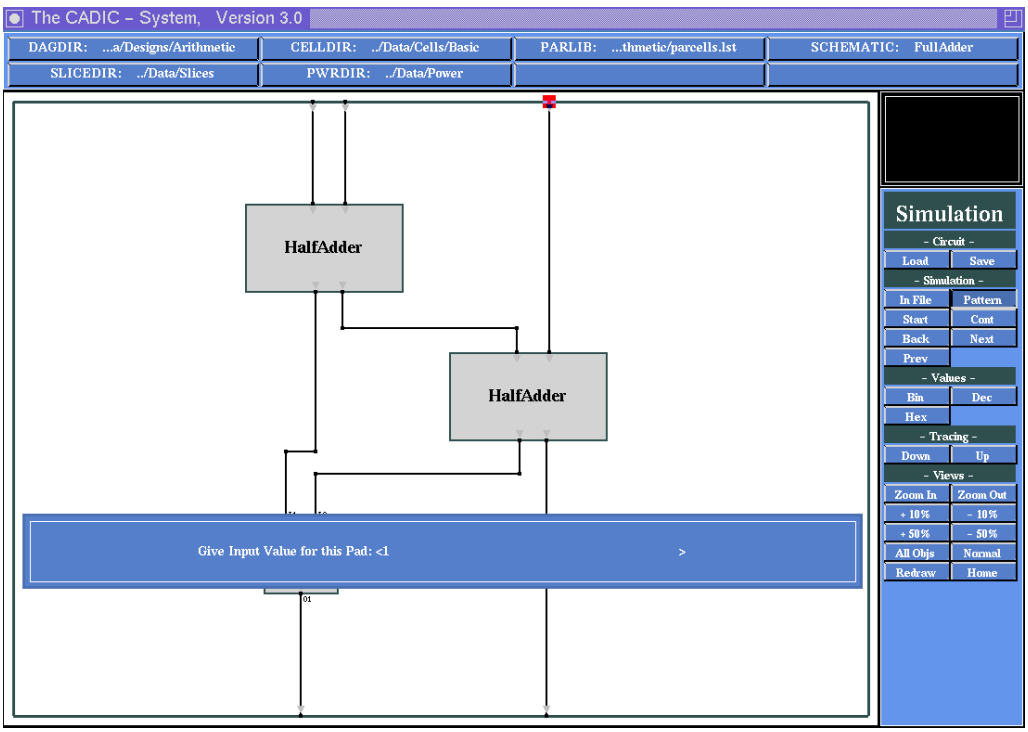


Figure 6.5: Setting input value for the right input pad

After pressing the return key, you will see a label beside the input pad, containing the binary input string. In the same way you have to specify the values for the middle and the left input pad. Here we enter the values `0` and `1` respectively, i.e. we want to check our fulladder design for the operation $1 + 0 + 1$. The pads are chosen according to the graphical input of the schematic, i.e. the wire which is connected to the border as first, is prompted first, etc.

A short time after you have input the last value, you will see the simulation results at the input and output pins of the macro cells. At each input

there exists a label containing a binary string, representing the values of the correspondig signals as it is shown in figure 6.6.

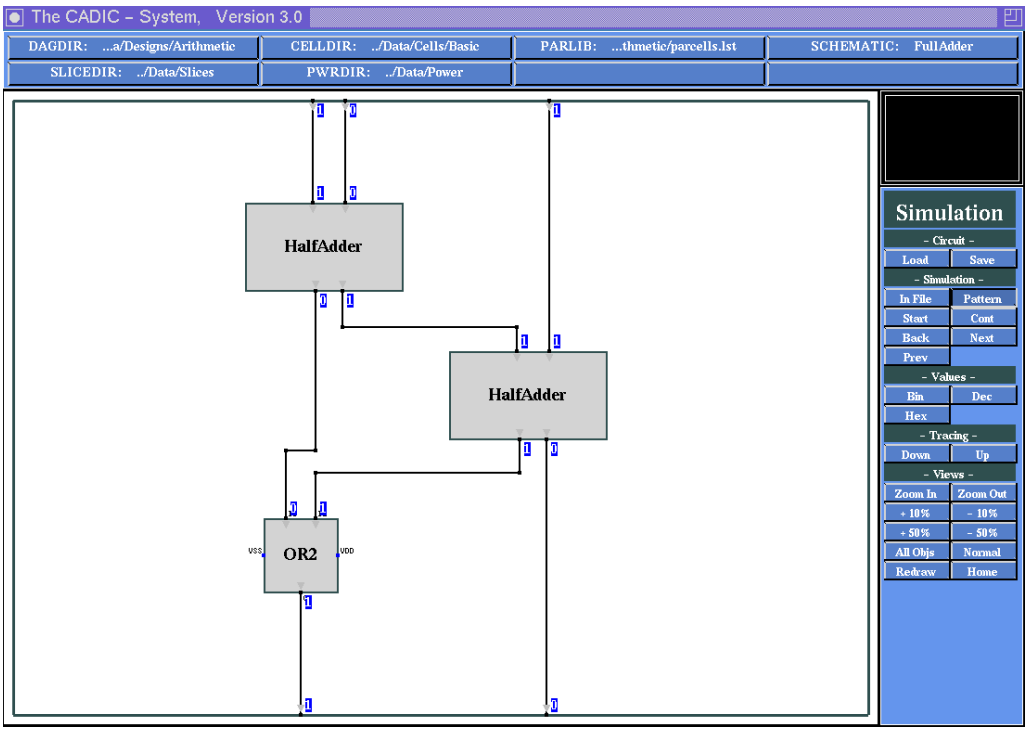


Figure 6.6: Simulation results for a single input pattern $1 + 0 + 1$

At the output signals of the circuit you see the values `1` and `0`, representing the carry value 1 and the sum value 0, i.e. for the input values 1, 0 and 1 the circuit computes the correct output values.

6.3.3 Tracing through Simulation Results

Now we want to examine the simulation results in detail. This can be done with the help of the tracing operations, which have already been explained in chapter 5. First we want to descend in the circuit hierarchy and control the simulation results at the next level. To do this select the entry `Down` from the submenu `-Tracing-`. Now you are in cell selection mode. Move the pointer into the workarea near to the upper instance `HalfAdder`. If it is highlighted in blue, press the left mouse button and descend into this macro cell. Now the schematic for `HalfAdder` is displayed and the simulation

results are shown at the input and output pins of the cells on this hierarchy level (c.f. figure 6.7).

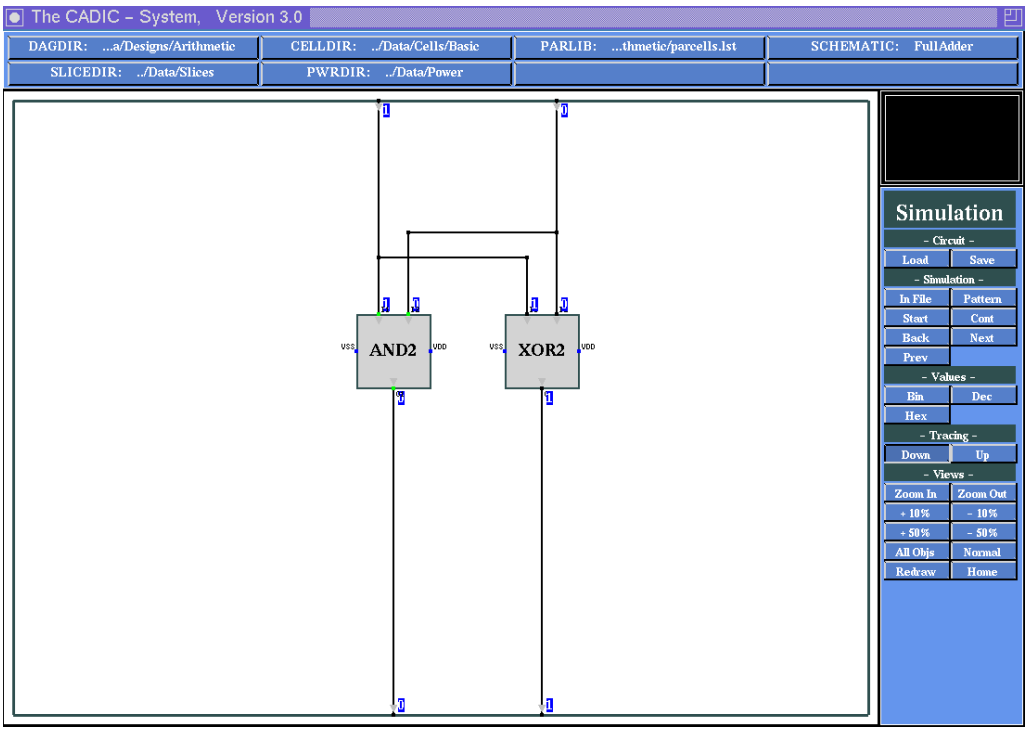


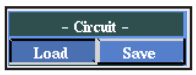
Figure 6.7: Simulation results at the `HalfAdder` level of the `FullAdder` specification

You can control the operation of the basic gates by comparing the input values to the generated output value. The task of `HalfAdder` is to compute the carry and sum of two single bits as we have shown in chapter 4. The first output value is the carry value, the second output is the sum. In this case the carry is `0` and the sum value is `1`, because we add `1 + 0`.

At the lowest level you terminate the trace down operation by pressing the right mouse button within the workarea. Now we will climb up the hierarchy to the topmost level in order to explain some other functions of the simulation tool. To do this move the pointer onto the entry `Up` in the submenu `-Tracing-` and press the left mouse button to reach the highest hierarchy level `FullAdder` (the current level name is shown in the upper right corner of the environment window).

6.4 More Simulation Functions

6.4.1 Load and Prepare for Simulation



In order to explain more functions of the simulation tool we will now examine the 4 bit conditional sum adder as a small example for a paramterized circuit. First we have to load and prepare this circuit for simulation. As shown above you can do this by selection of the entry `Load` from the submenu `-Circuit-`. From the schematic list window you should select the entry `CSADDER[n]`. We choose this schematic because it contains the additional wiring subcircuits which help us to check the simulation results very easily as it will become clear in the following.

Because this is a parameterized design, you will be prompted to enter the values for its parameters. After specifying these values the system will be able to build the hierarchical data structure as it is described in chapter 5.

In our case we choose $n = 4$, i.e. simply type `4` to the prompt window

```
Enter Parameter Values for "CSADDER[n]":  <4    >
```

and press the return key. In the message window you will see the hierarchy levels which are allocated for `CSADDER[4]`.

Now `CSADDER[4]` is loaded and it will be prepared for simulation. The system will calculate the data flow through the circuit and will generate a C module for the execution of the circuit operations (c.f. figure 6.3).

This module will be created in the directory given by the environment variable `SIMDIR`. In the message area you can see the following information:

```
cannot open file ../Data/Sim/CSADDER[4].out
```

This shows that the executable simulation program does not yet exist. During the creation of the executable simulation program the system will display the message

```
Generation of the simulation program:
```

In the workarea the circuit is not fully shown, until the process of translating

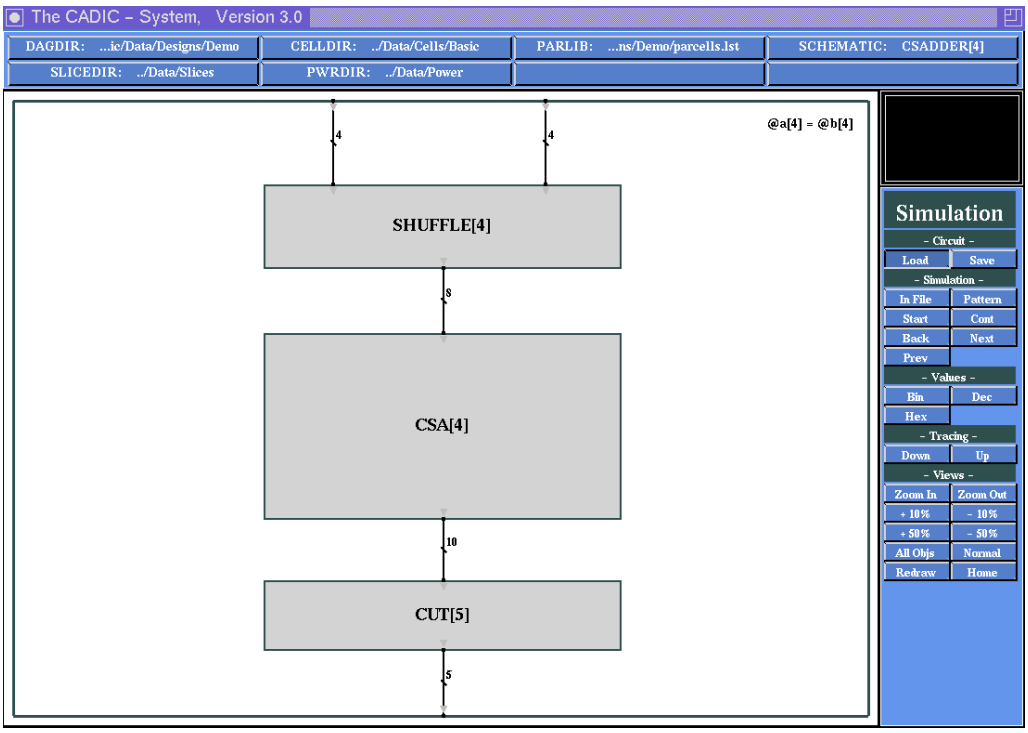


Figure 6.8: Circuit `CSADDER[4]` loaded and prepared for simulation

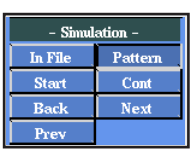
and linking of the simulation program has terminated. If the system displays the message

```
--- Finished
```

the executable file is created and the system is ready for starting the circuit simulation. At this moment your workarea should look like that shown in figure 6.8.

In figure 6.8 you will see that the system has calculated the data flow through the circuit. This is indicated by the small arrows at the inputs and outputs of the macro cells.

6.4.2 Simulation of a Single Pattern



Again we want to simulate the circuit for a single input pattern first. For this purpose select the entry `Pattern` from the `-Simulation-` submenu. After activating this function the system asks you for the values at the input pads

of the circuit. Your screen should look like that shown in figure 6.9. First you have to enter the input value for the left pad at the northern border of `CSADDER[4]` which is highlighted by a small square. At the prompt window you can give a binary input value for this pad. Just type in

```
Give Input Value for this Pad <11        >
```

as it is shown in figure 6.9. Note that the binary string must have a length of 4, because of the width of the corresponding input wire. For this reason the system will fill up your input with leading zeros.

If you enter more digits than given by the wire width, your input will be refused and the prompt window will popup again.

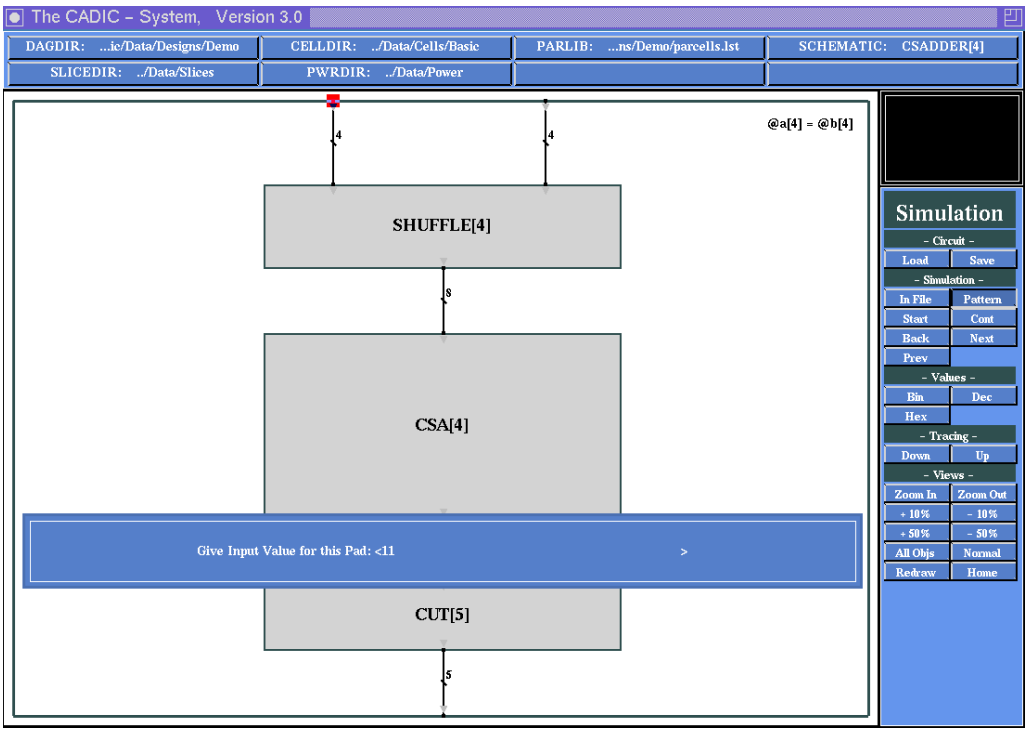


Figure 6.9: Setting input value for the left input pad

After pressing the return key, you will see a label beside the input pad, containing the binary input string. In the same way you have to specify the value for the right input pad. Here we enter the value `10`, i.e. we want to check our conditional sum adder design for the operation $11 + 10$.

A short time after you have input the last value, you will see the simulation results at the input and output pins of the macro cells. At each input there exists a label containing a binary string, representing the values of the corresponding signals as it is shown in figure 6.10.

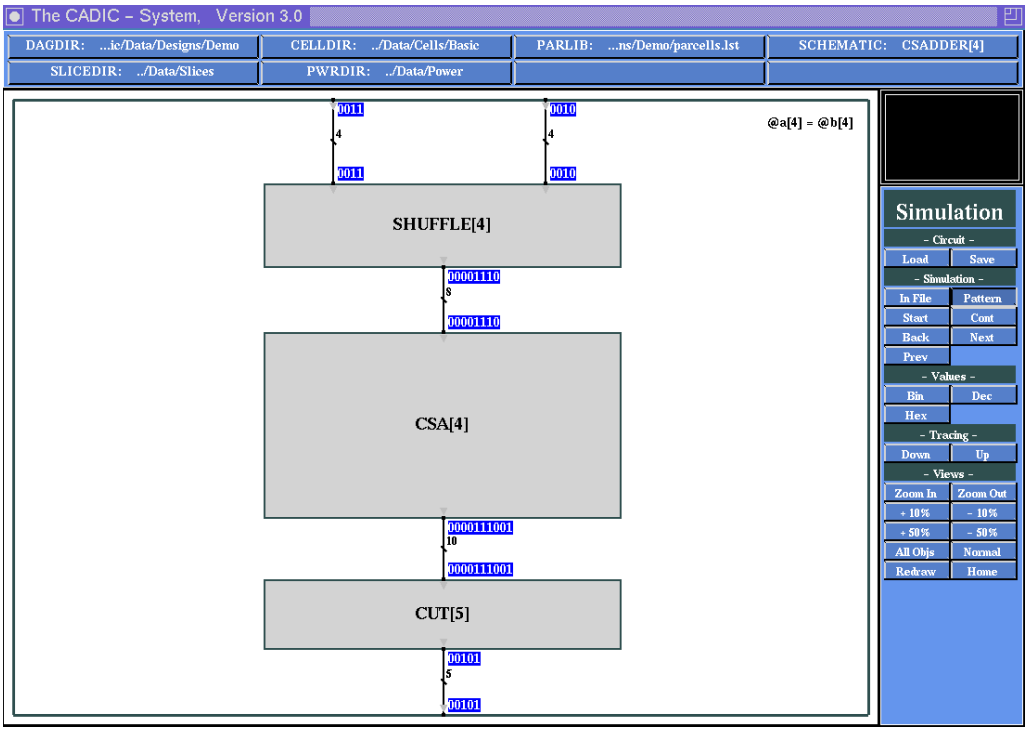
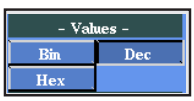


Figure 6.10: Simulation results for a single input pattern 11 + 10

At the output signal of the circuit you see the value `00101`, representing the decimal value 5, i.e. for the input values `11` and `10` the circuit computes the correct output values. Again you can check the simulation results at the lower hierarchy levels with the help of the tracing operations. If you trace through the hierarchy you should return to the highest level before you continue with the following operations.

6.4.3 Changing the Display Mode



Currently the input and output values are displayed as binary strings, i.e. as sequences of zeros and ones. The length of each string is given by the width of the corresponding wire. In some cases (e.g. arithmetic circuits) it is more

convenient to interprete these binary strings and display them according to a certain base. Here we offer the modes decimal and hexadecimal. Now we change the display mode to decimal by pressing the left mouse button onto the entry `Dec` in the submenu `-Values-`. The bitstrings are converted to decimal numbers and you see the input values (`3`, `2`) and the output value `5`. Note that now the input and output is done in decimal mode .

We control this by entering a new single pattern for the simulation. Move the pointer onto the entry `Pattern` in the submenu `-Simulation-`. After pressing the left mouse button you are asked for the input value of the left input pad (c.f. figure 6.9). Now you can type in a decimal value, e.g. `9`. Note that the binary representation of the decimal value must not exceed the width of the corresponding wire. In our case of `CSADDER[4]`, where the input wire have width 4, you may input numbers in the range from `0` to `15`. for the second input pad enter the value `4` and press the return key.

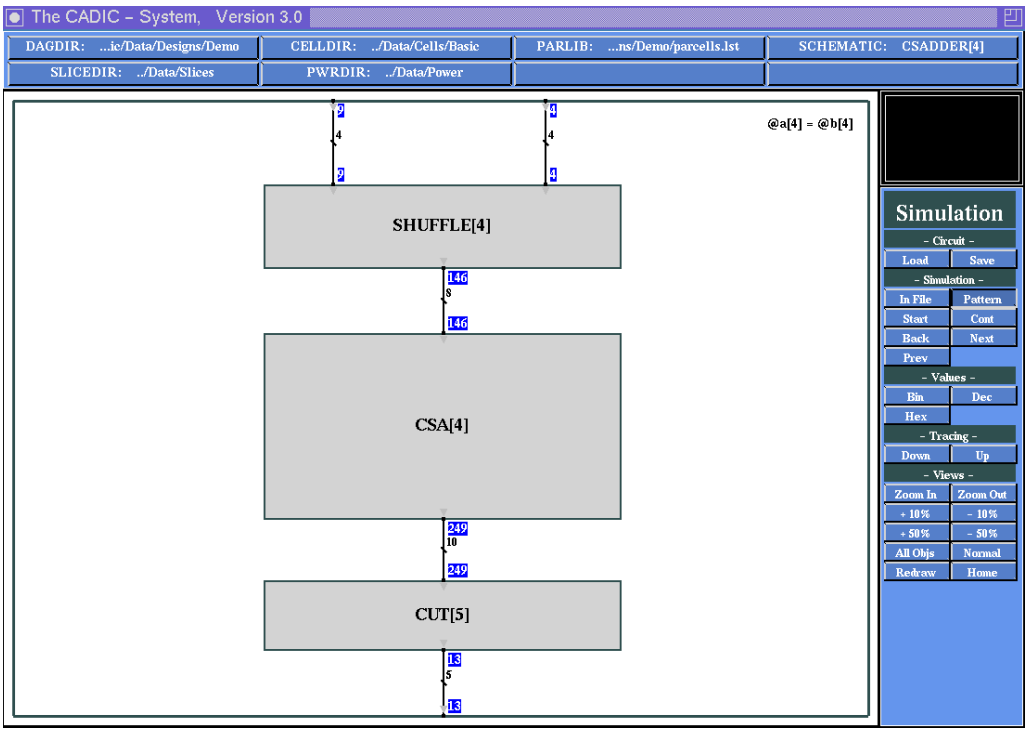


Figure 6.11: Display of simulation results in decimal mode

After the simulation has been finished you see the results at the input and

output pins of the macros which are all displayed in decimal mode (c.f. figure 6.11). The result of the addition is `13` which shows that our adder works correct for this input example.

The decimal display mode is very intuitive for controlling simulation results at a high level of the circuit hierarchy. When you descend in the hierarchy and the wires are more and more split into single binary values, it is more convenient to use the binary display mode.

6.4.4 Simulation of a List of Patterns

The simulation of a single pattern is useful, if you want to examine the behaviour of your design for special input values. But in general, you have to simulate it for a lot of patterns. Then it would be very tiresome to input all patterns in single pattern mode. The simulation of a list of patterns can easily be done with the help of the function `In File` from the `-Simulation-` submenu. After activating this mode, a window will be popped up onto the workarea. This window contains the system call

```
vi ../Data/Sim/CSADDER[4].pat
```

i.e. it calls the `vi` editor with the file `CSADDER[4].pat`, on which you can write simulation patterns for the execution by the simulation tool. This file is also located in the directory given by the environment variable `SIMDIR`. At this point of our session the file does not exist and in the bottom line of the new window you can read the message

```
"../Data/Sim/CSADDER[4].pat" [New File]
```

which is the normal message of `vi` after opening a new file. In the following we suppose that you are familiar with the work with `vi`.

Now type in the following text:

```
DAG CSADDER[4];
PAR 4;
```

```
INPUT n[0:3], n[4:7];
OUTPUT s[0:4];
PATTERNS
00000000
00000
00000001
00001
00000010
00010
00010000
00001
00100000
00010
00100010
00100
00110011
00101
01000100
01000
01000101
010001
```

After you have closed the `vi` window you can start the simulation of the pattern list. For this purpose select the entry `Start` from the submenu `-Simulation-`. In the message area you can see, how many patterns have been simulated successfully. Each pattern is simulated and compared to the expected output in the pattern list. If there is a difference between the simulation result and the expected output, the simulation stops and the current simulation results are displayed at the input and output pins of the macro cells.

In this case there is a mistake on our pattern file. Therefore the simulation will stop when pattern 7 is simulated. In the message area the following

information is given

```
Wrong Simulation Values for pattern 7

Simulation Output:  00110011 0 0 00110

 Specified Output:  00110011 0 0 00101
```

The second line shows the simulation results for pattern 7 and the third line shows the expected results. These are grouped according to the borders of the circuit, beginning with the northern border, on which the inputs are located, i.e. input string is `00110011`. The next two elements are at the eastern and western border. These have no input or output pads, and their default string is `0`. The last string is located at the southern border which is the output of the circuit. Here is the difference between the simulation result `00110` and the expected result `00101`.

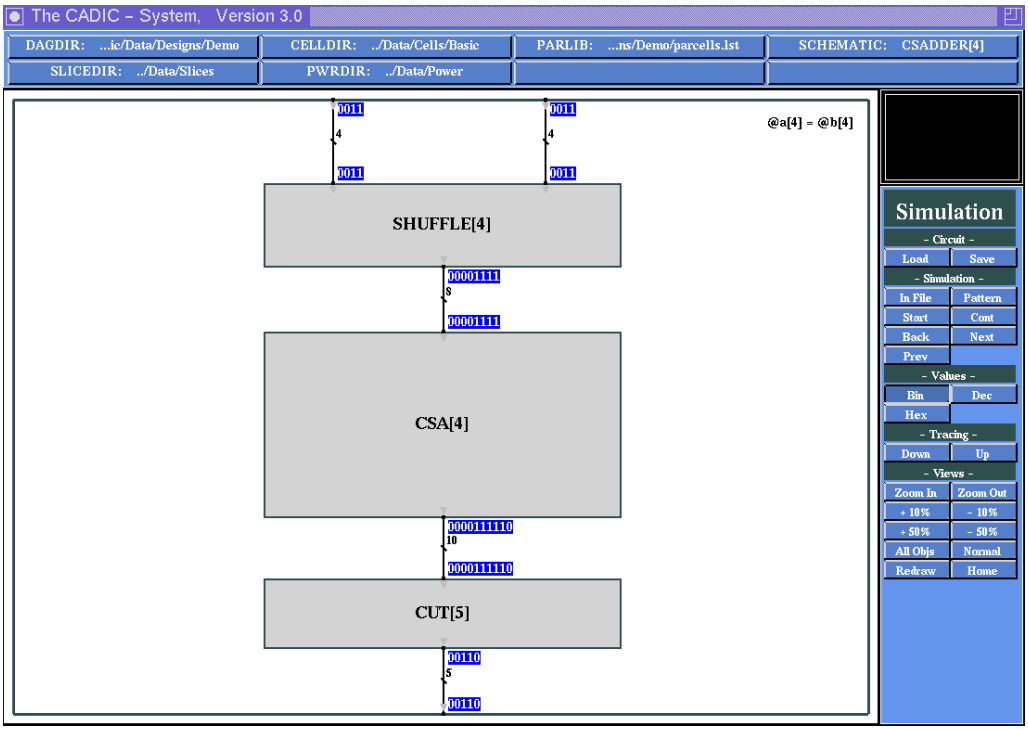
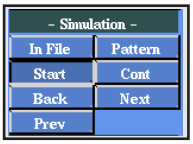


Figure 6.12: Simulation results for pattern 7

If you notice a difference between the simualtion result and the expected result, you can now examine your design with the help of the tracing func-

tions and locate the error. In our case it is not a design error, but we have a wrong entry in our pattern list. The inputs to our adder are namely `0011` and `0011`, i.e. the decimal numbers $(3, 3)$. The result of this addition should be 6, i.e. `00110` in binary representation. We have to change the corresponding line on our pattern file which can be done by calling the function `In File` from the submenu `-Simulation-` again.



Change the pattern file in the appropriate way and save it into `SIMDIR`. After that you can select the entry `Start` from the submenu `-Simulation-` again. Finally you will see the messages

`Simulation successful`

in the message area. This means that all patterns have been sucessfully simulated and the simulation loop has reached the end of the pattern file.

If you want to check the patterns step by step you can first reset the simulation by selecting the entry `Back` from the submenu `-Simulation-`. With the help of this function you can return to a previously created wrong simulation result. In our case there is no such input pattern anymore and you return to the beginning of the pattern list. In the message area you can read the information

`No previous error found.`

Now you can step through the patterns in your list. For each pattern the simulation results are shown at the input and output pins of the macro cells. With the help of the tracing functions you can examine the design for each simulation pattern. If there is no difference between the simulation result and the expected result of the current pattern, you will see the message

`Simulation sucessful`

in the message area. If you repeat this function and reach the end of the pattern list, the system will display the message

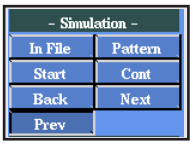
`Simulation terminated`

and the labels at the input and output pins will turn empty. Any further

activation of `Next` will result in the messages

```
Simulation terminated
```

as we have shown it above. This tells you that the end of the pattern list is reached. You can now step backwards with the help of the following function or restart the simulation by selecting the entry `Start`.



In the same way as you can step forward through the list of patterns you can go backwards to previous patterns. This can be done by selecting the entry `Prev` form the submenu `-Simulation-`.