

5

Arithmetics

This chapter introduces basic concepts concerning the design of arithmetic gates. The adder circuit is presented, with its corresponding layout created manually and automatically. Then the comparator, multiplier and the arithmetic and logic unit are also discussed. This chapter also includes details on a student project concerning the design of binary-to-decimal addition and display.

Half-Adder Gate

The Half-Adder gate truth-table and schematic diagram are shown in Figure 5-1. The SUM function is made with an XOR gate, the Carry function is a simple AND gate.

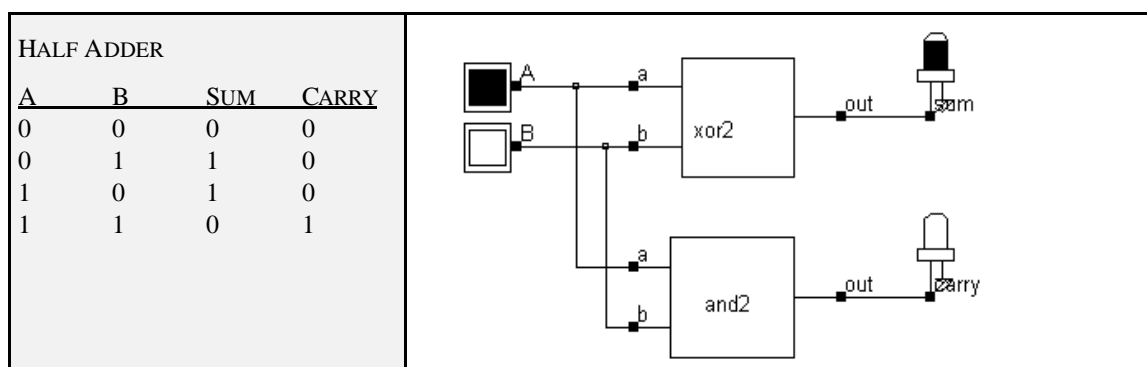


Fig. 5-1. Truth table and schematic diagram of the half-adder gate (HADD.MSK).

FULL CUSTOM LAYOUT	You may create the layout of the half-adder fully by hand in order to create a compact design. Use the polysilicon and metal1 layers for short connections only, because of the high resistance of these materials. Use Poly/Metal, Diff/Metal contact macros situated in the upper part of the Palette menu to link the layers together.
LAYOUT LIBRARY	Load the layout design of the Half-Adder through the File -> Open and HADD.MSK sequence.
VERILOG COMPILING	1. Use DSCH2 to create the schematic diagram of the half-adder. Verify the circuit with buttons and lamps. Save the design under the name 'hadd.sch'

	<p>using the command File -> Save As.</p> <ol style="list-style-type: none"> 2. Generate the Verilog text by using the command File -> Make Verilog File. 3. In Microwind2, click on the command Compile -> Compile Verilog File 4. Select the text file 'hadd.txt'. <pre> module Hadd(B,A,sum,carry); input B,A; output sum,carry; xor xor1(sum,B,A); and and1(carry,A,B); endmodule </pre> <ol style="list-style-type: none"> 5. Click on Compile. When the compiling is complete, the resulting layout appears shown below. The XOR gate is routed on the left and the AND gate is routed on the right 6. Click on Simulate ->Start Simulation. The timing diagrams of figure xxx appear and you should verify the truth table of the half-adder. Click on Close to return to the editor.
--	--

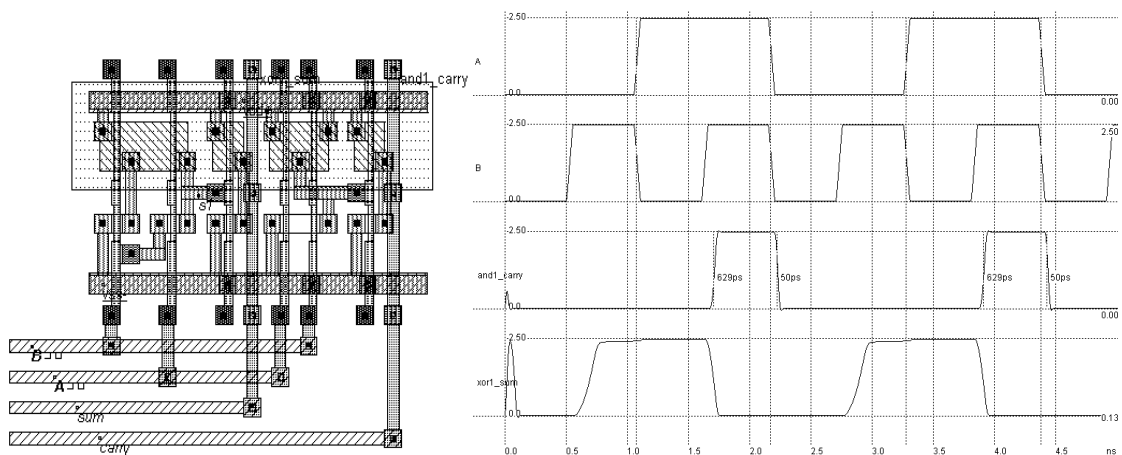


Fig. 5-2. Compiling and simulation of the half-adder gate (Hadd.MSK)

Full-Adder Gate

The truth table and schematic diagram for the full-adder are shown in Figure 5-3. The SUM is made with two XOR gates and the CARRY is a combination of NAND gates, as shown below. The most straightforward implementation of the CARRY cell is $AB+BC+AC$. The weakness of such a circuit is the use of positive logic gates, leading to multiple stages. A more efficient circuit consists in the same function but with inverting gates.

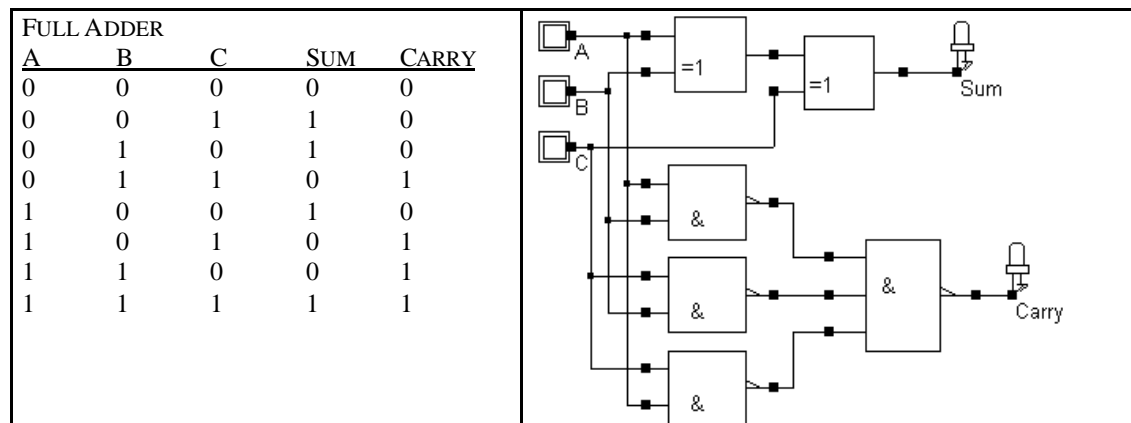


Fig. 5-3. The truth table and schematic diagram of a full-adder(FADD.SCH)

Full-Adder Symbol

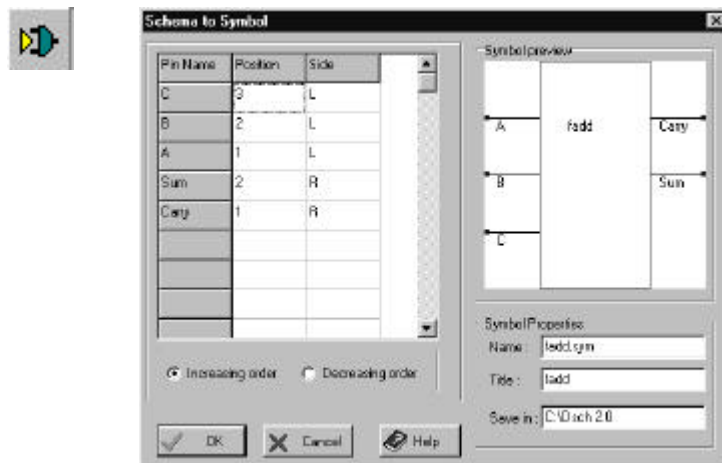


Fig. 5-4. Create a symbol from the schematic diagram

In order to build hierarchical designs using the adder, we detail the procedure to generate the symbol of the full-adder from its schematic diagram. In DSCH2, click the above icon, the screen of the right hand side appears. Simply click 'OK'. The symbol of the full-adder is created, with the name 'Fadd.sym' in the current directory. Use the command "Insert -> Symbol" to include this symbol into a new circuit. For example, the circuit 'FaddTest' includes the hierarchical symbol and verifies its behavior.

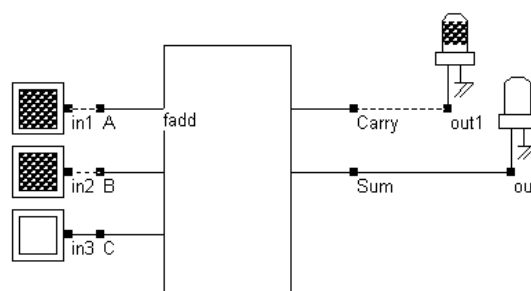


Fig. 5-5. Testing the new ADDER symbol(FaddTest.SCH)

Full-custom design of the adder

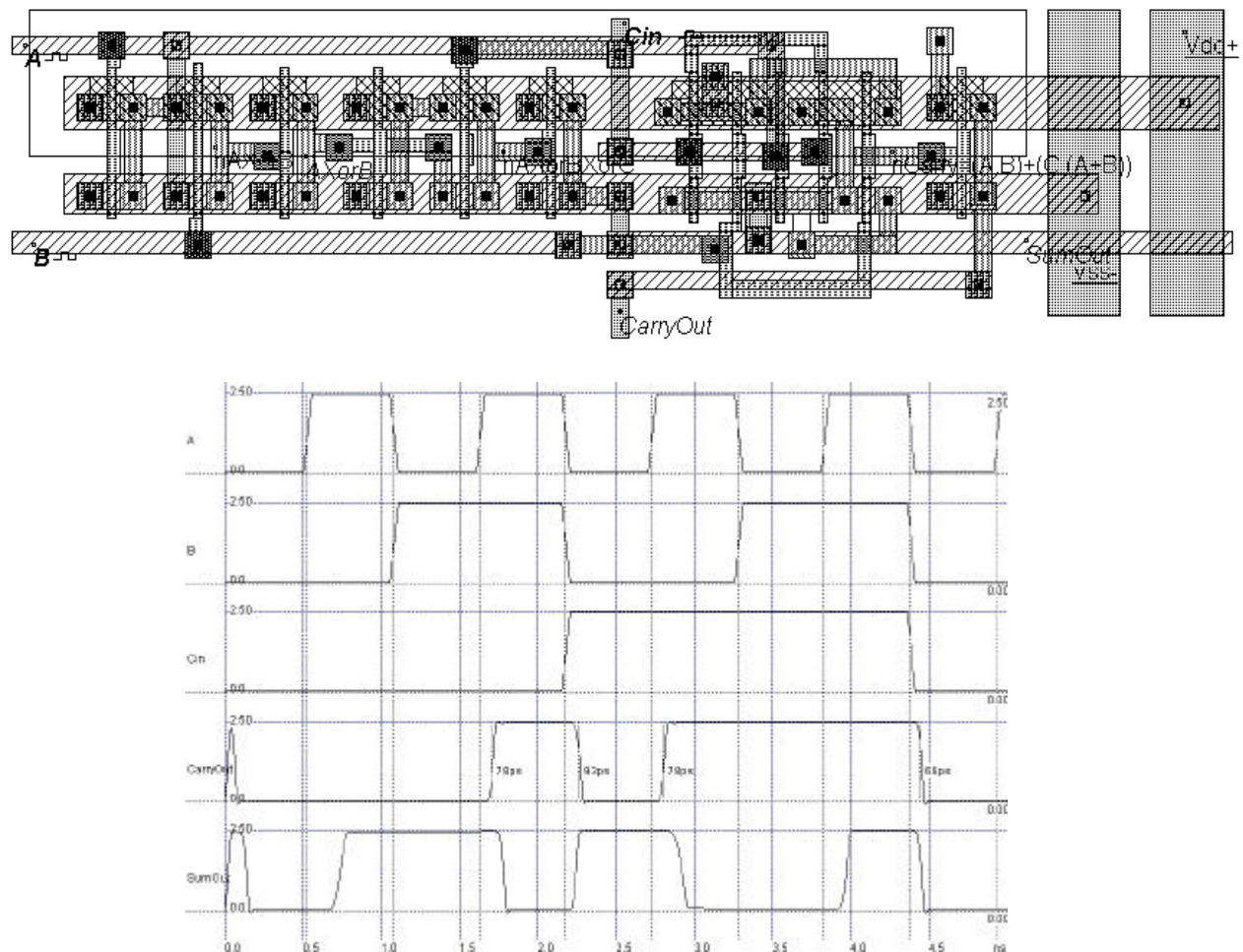


Fig. 5-6. The full-custom implementation of the full-adder and its simulation (FullADD.MSK).

<p>FULL CUSTOM LAYOUT</p>	<p>You may create the layout of the full-adder by hand in order to create a compact design. Notice that the AND/OR combination of cells may be replaced by a complex gate. An example of full-custom layout of the full-adder is proposed in Figure 5-6. Notice that the carry propagates vertically within the cell to ease multiple addition. The typical delay is less than 100ps in 0.25μm technology.</p>
<p>VERILOG COMPILING</p>	<ol style="list-style-type: none"> 1. Use DSCH2 to create the schematic diagram of the full-adder. Verify the circuit with buttons and lamps. Save the design under the name 'fadd.sch' using the command File -> Save As. 2. Generate the Verilog text by using the command File -> Make Verilog File. 3. In Microwind2, click on the command Compile -> Compile Verilog File 4. Select the text file 'fadd.txt'.

```

module fulladd(sum,carry,a,b,c);
  input a,b,c;
  output sum,carry;
  wire sum1;

  xor xor1(sum1,a,b);
  xor xor2(sum,sum1,c);
  and and1(c1,a,b);
  and and2(c2,b,c);
  and and3(c3,a,c);
  or or1(carry,c1,c2,c3);
endmodule

```

5. Click on **Compile**. When the compiling is complete, the resulting layout appears shown below. The XOR gate is routed on the left and the AND gate is routed on the right
6. Click on **Simulate ->Start Simulation**. The timing diagrams of figure xxx appear and you should verify the truth table of the half-adder. Click on **Close** to return to the editor.

The simulation of the full-Adder is conducted in figure 5-7, and exhibits propagation delays twice higher than the full-custom design. Consequently, the Verilog translation is appropriate for

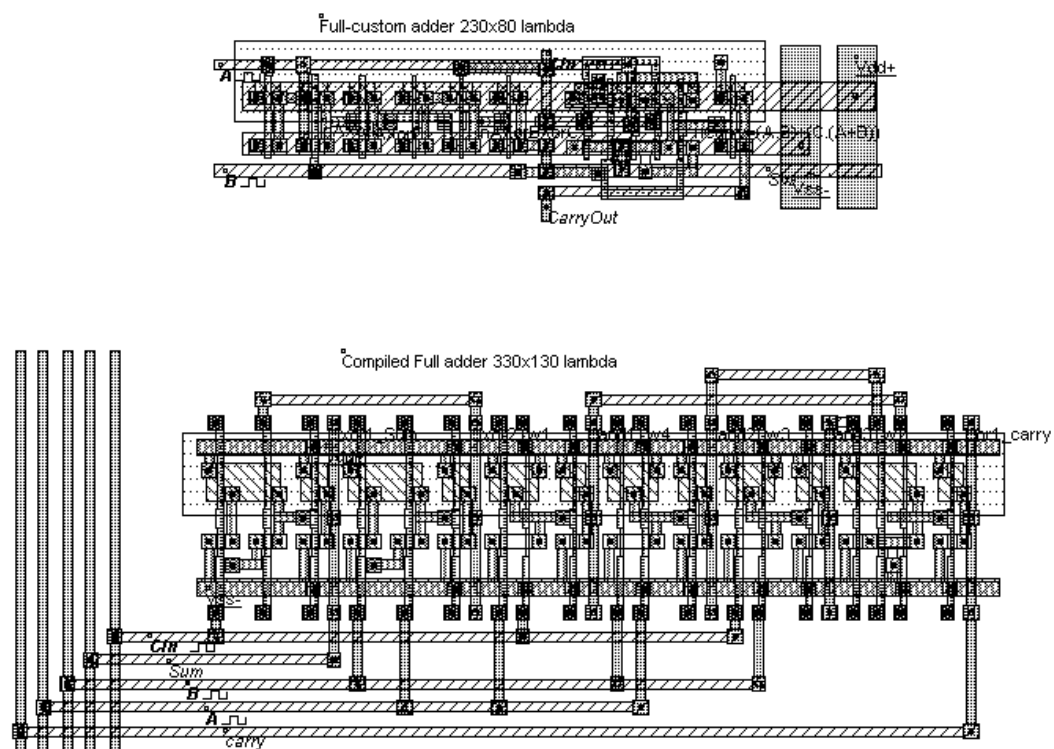


Fig. 5-7. Comparison between full-custom and compiled layout of the full-adder

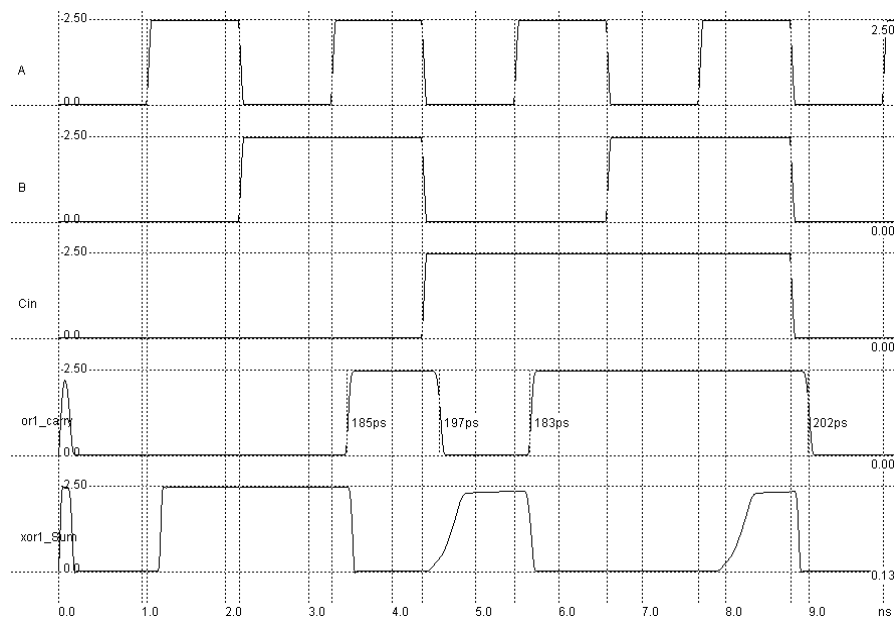


Fig. 5-8. Simulation of a full-adder (File FADD.MSK).

Four-Bit Adder

This circuit includes full-adders in serial, so that the result of each stage propagates to the next one, from the top to the bottom. The circuit allows a four-bit addition between two numbers A3,A2,A1,A0 and B3,B2,B1,B0. Insert the user-defined 'Fadd.sym' symbol using the command **Insert -> User Symbol**. In DSCH2, the A and B numbers are generated by keyboard symbols, as reported below. Also notice the hexadecimal display with a ground connected to the K input to activate the display.

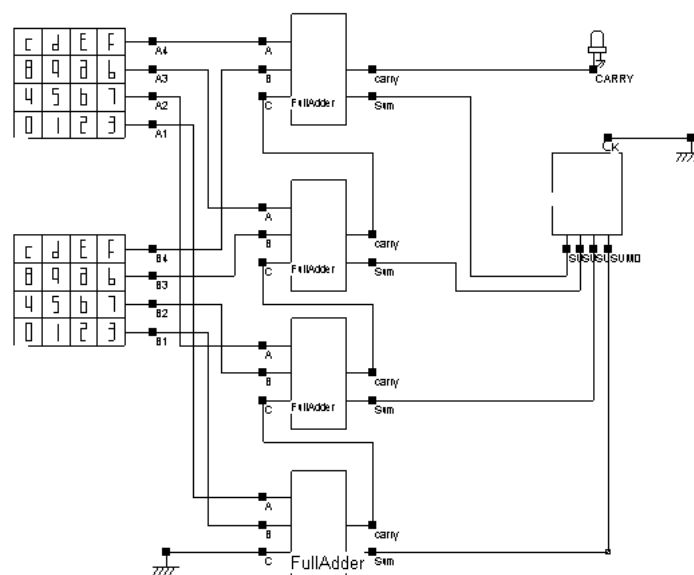


Fig. 5-9. Schematic diagram of the four-bit adder (ADD4.SCH).

Figure xxx details the four-bit adder layout based on the full-custom cell design, with the corresponding simulation. In Microwind2, the command **Edit -> Duplicate X,Y** has been used to duplicate the full-adder layout vertically.

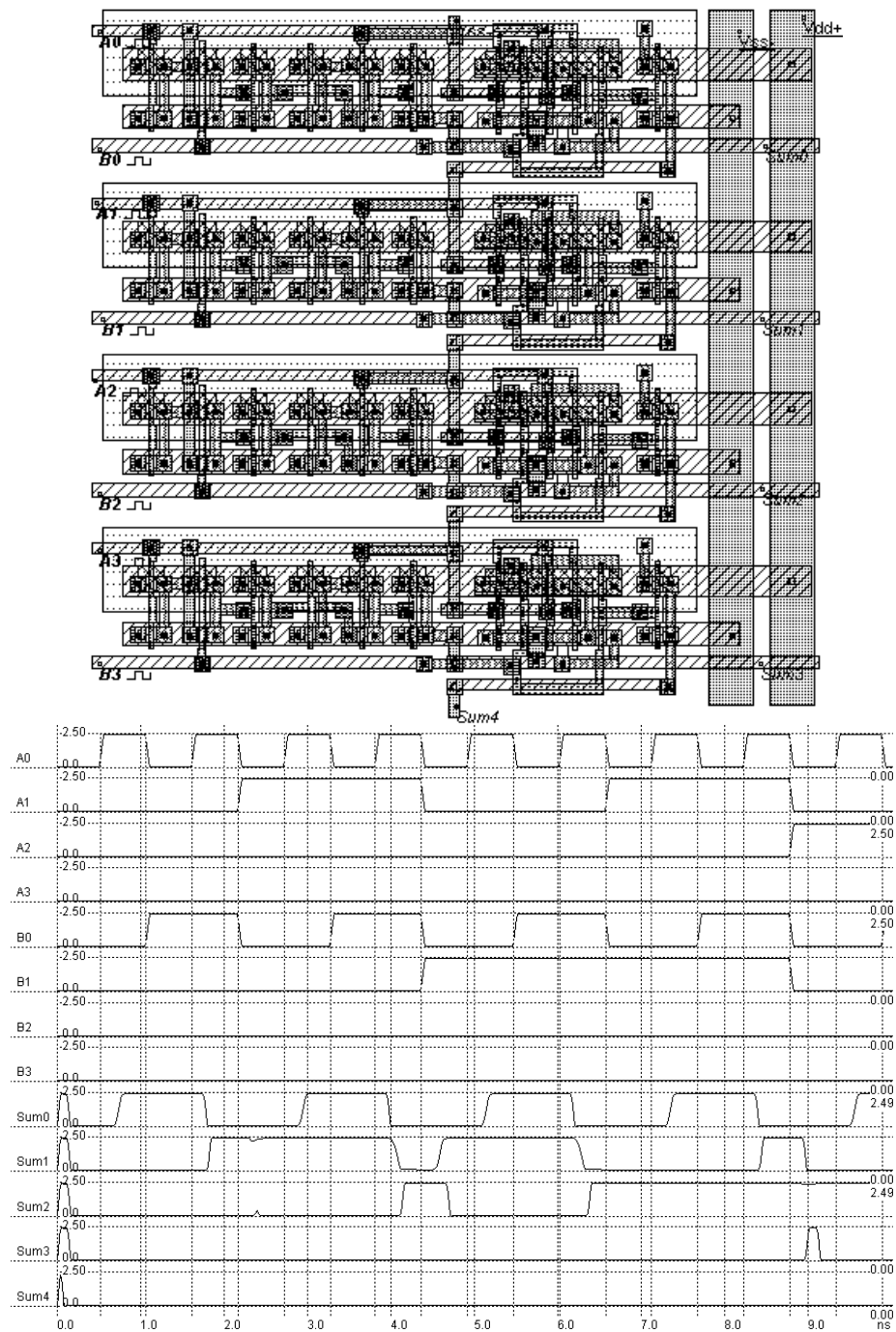


Fig. 5-10. Design and simulation of the four-bit adder (ADD4.MSK).

Comparator

The truth table and the schematic diagram of the comparator are given below. The $A=B$ equality represents an XNOR gate, and $A>B$, $A<B$ are operators obtained by using inverters and AND gates.

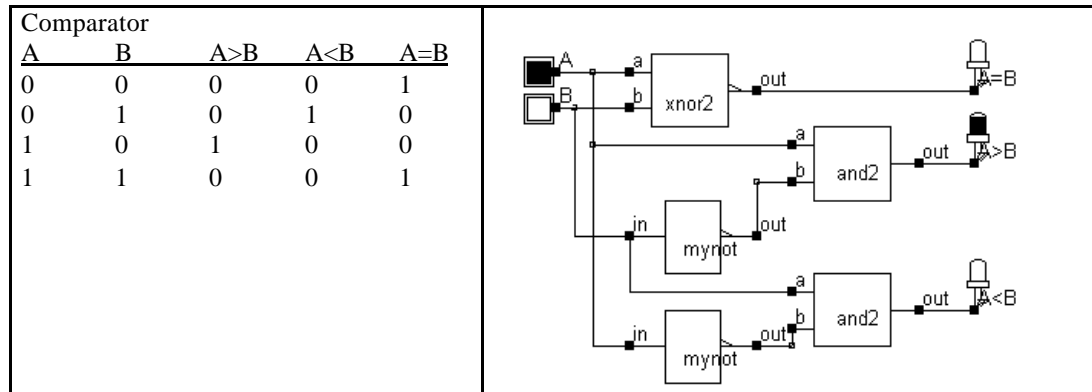
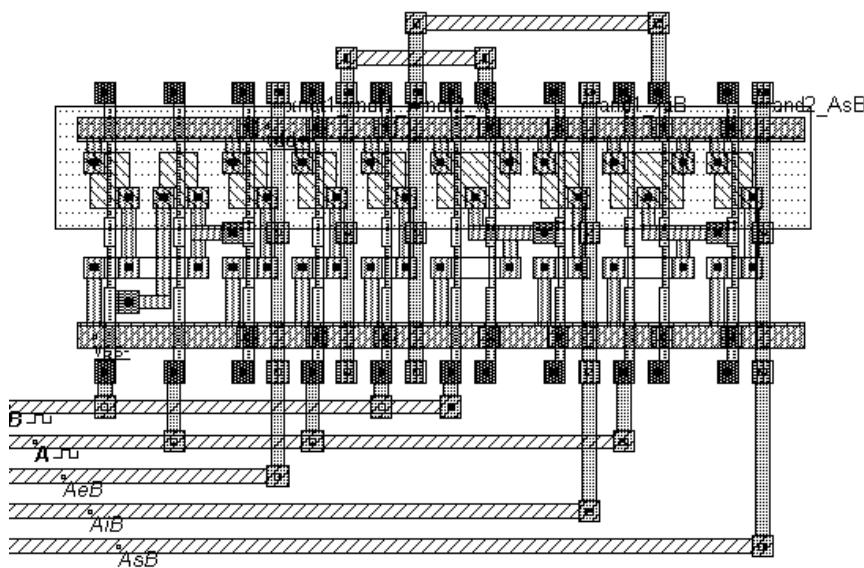


Fig. 5-11. The truth table and schematic diagram of the comparator (COMP.SCH).

Using DSCH2, the logic circuit of the comparator is designed and verified at logic level. Then the conversion into Verilog is invoked (**File -> Make verilog File**). Microwind2 compiles the verilog text into layout. The layout and simulation of the comparator is given in Figure xxx. The XNOR gate is located at the left side of the design. The inverter and NOR gates are at the right side. After the initialization, $A=B$ rises to 1. The clocks A and B produce the combinations 00,01,10 and 11. Notice the small glitch on $A>B$ at $t=xxx$ ns. This glitch is not a design error. On the contrary, it shows that during the transition of A and B the situation $A>B$ occurs and that the cell is fast enough to react.



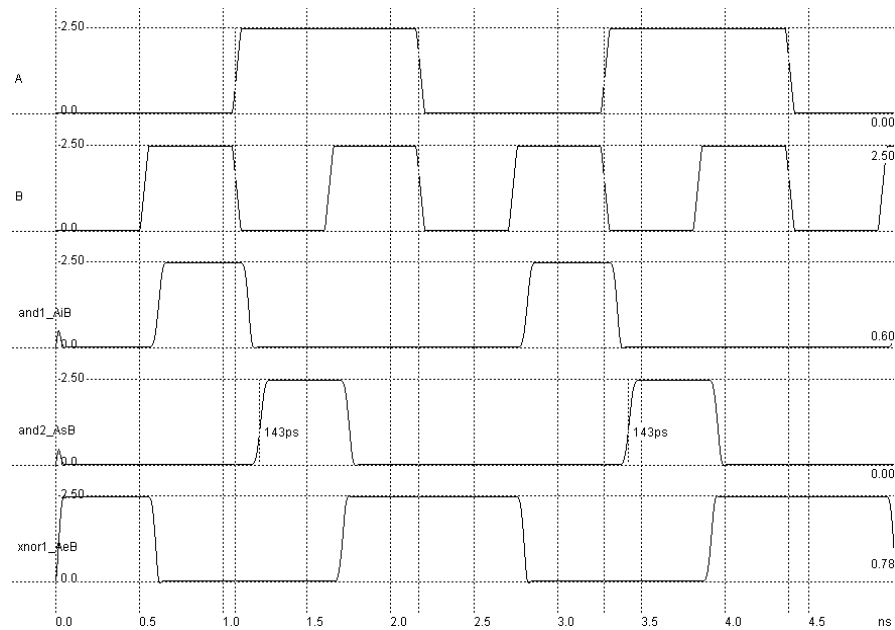


Fig. 5-12. Simulation of a comparator (COMP.MSK file).

n-bit Comparator

The technique for n-bit comparison is based on the use of adder circuits. In the schematic diagram of figure xxx, the adder system is modified into a comparison system, by computing the Boolean function close from $A-B$. The 'equal' operator is built using simple and functions. Consequently, the one-bit comparator includes the full-adder layout, one inverter and one NAND gate.

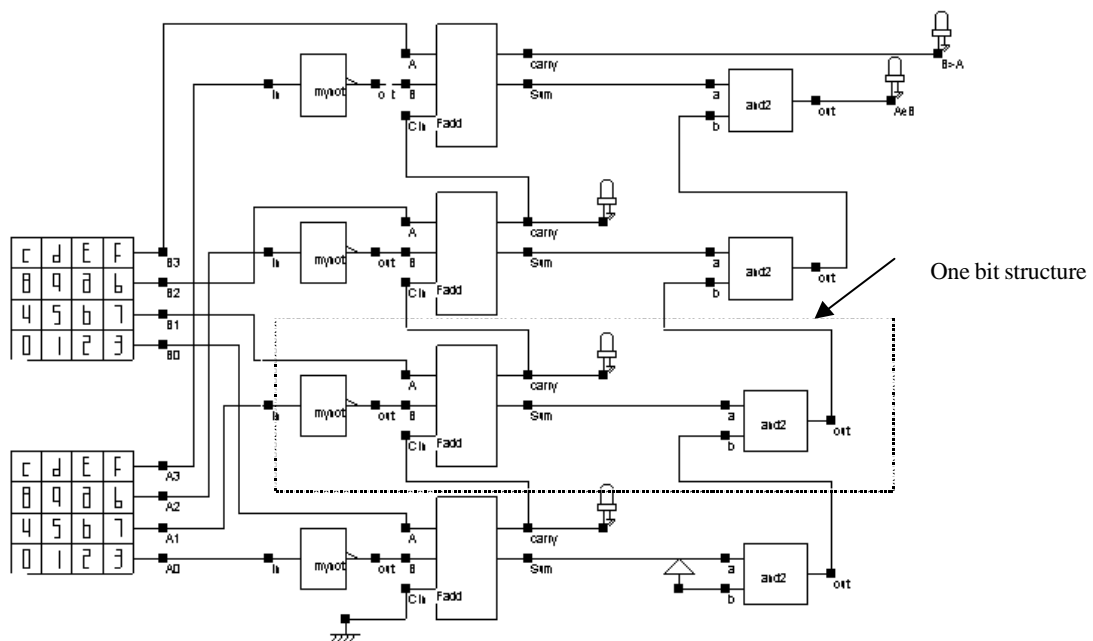
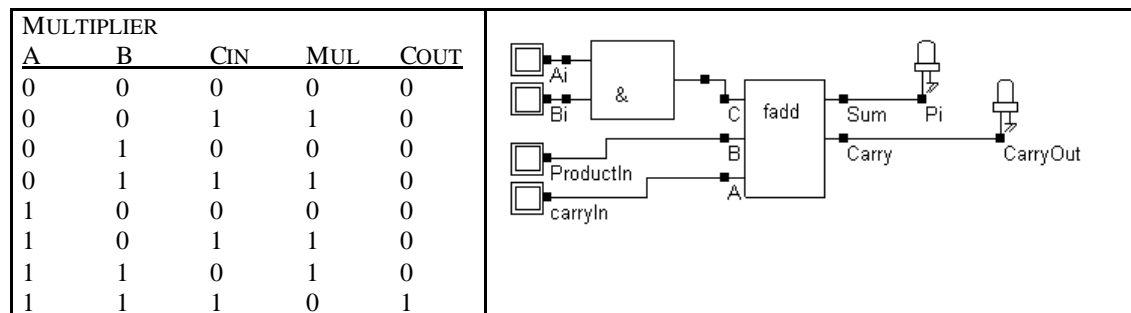


Fig. 5-13. Schematic diagram of a 4-bit comparator (COMP4.SCH).

Multiplier

The multiplication of integer numbers A and B can be implemented in a parallel way using elementary binary multiplication. The corresponding cell should verify the truth table given below. The cell can be made up of a full-adder cell and an AND gate, as shown in the schematic diagram below (MUL1.SCH).



A 4x4 bit multiplication is proposed in Figure 5-14. The circuit multiplies input A (Upper keyboard) with input B (Lower keyboard) which produces a result P, as detailed in Figure 5-14.

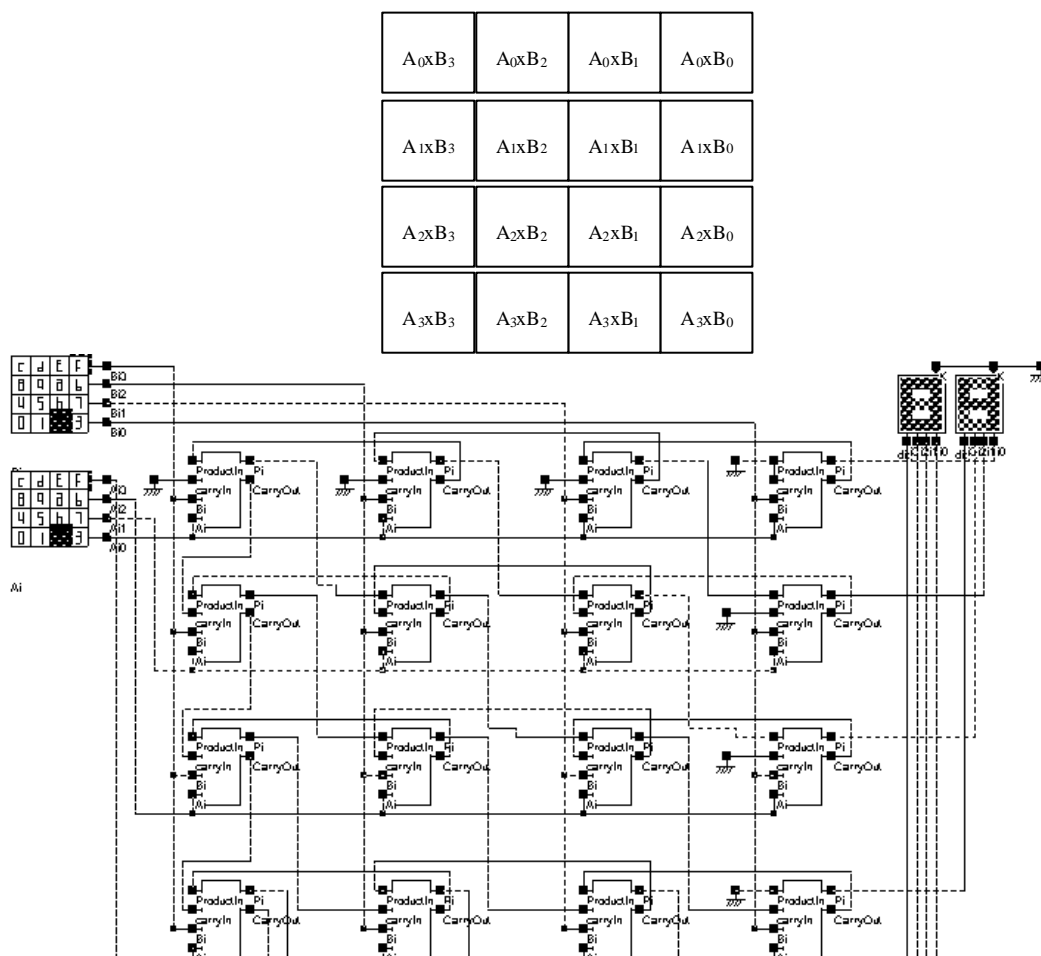


Fig. 5-14. Schematic diagram of the 4x4 bit multiplier (MUL4x4.SCH).

The cell is built for an iterative implementation. The inputs and outputs are organized in such a way that the multiplication array is regular. Let us illustrate the multiplication process through a small example (6 x 7=42). The basic mechanism is the addition, which involves a carry and the product $A_i.B_j$. The sum propagates down to the result. The key idea is to connect the carry to the cell situated at the left side. In the example below, when a 1 is added vertically to another 1, the sum goes one row below, while the carry is sent on the left cell.

$$\begin{array}{r}
 A \quad 0110 \quad (6) \\
 \underline{B \quad 0111 \quad (7)} \\
 0110 \\
 0110 \\
 0110 \\
 \underline{0000} \quad . \\
 00101010 \quad (42)
 \end{array}$$

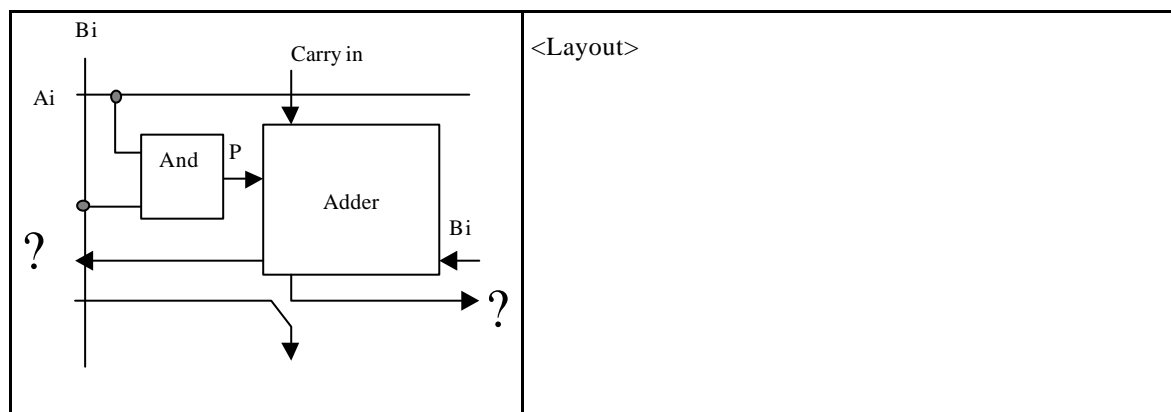


Fig. 5-15. Design of the 4x4 bit multiplier (MUL4x4.MSK).

Fig. 5-16. Simulation of the 4x4 bit multiplier (MUL4x4.MSK).

Arithmetic and logic Units (ALU)

The digital function that implements the micro-operations on the information stored in registers is commonly called an arithmetic logic unit (ALU). The ALU receives the information from the registers and performs a given operation as specified by the control.

A very simple ALU design is proposed to illustrate its principle. The control unit is made up of a 4-1 multiplexor. The operation part consists of four kinds of operations listed as follows: and, or, addition and subtraction. The 'and' and 'or' operation are realized by using the basic logic gates. The addition and subtraction are realized using the ADDER user symbols.

A digital multiplexer made from MOS devices selects one of the 4 operations results and directs it to a single output line « Result ».

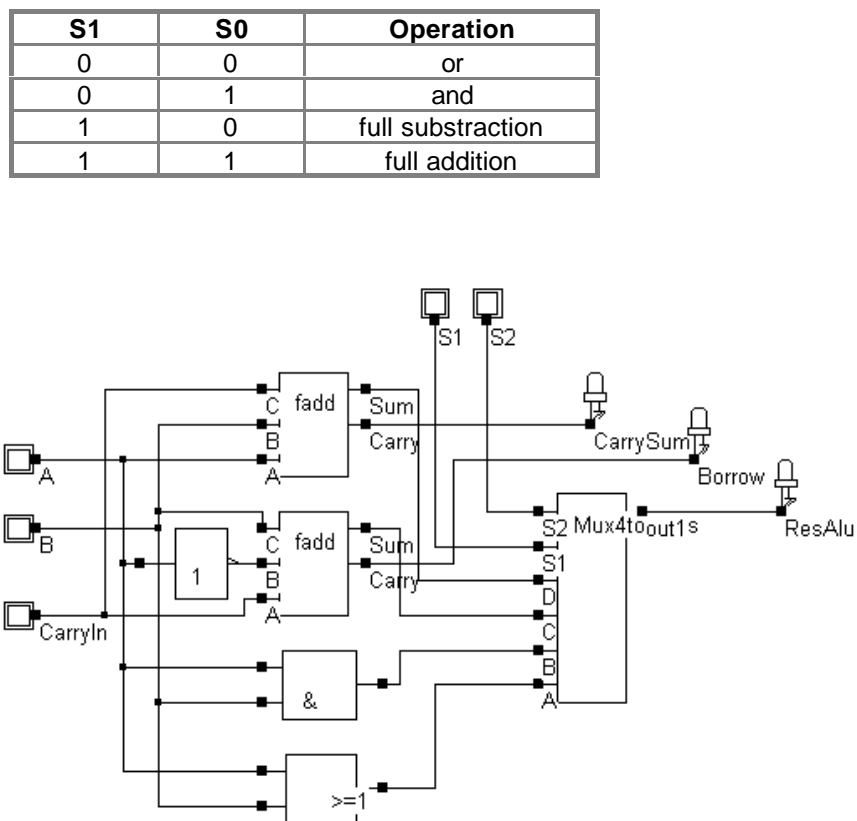


Fig. 5-17. The 1-bit ALU operates the and, or, addition and subtraction (ALU1bit.SCH)

PROJECT: A 4-BIT BCD ADDER

The objective of this project is to perform the addition of two BCD (Binary Decimal Code) numbers X and Y ranging from 0 to 9, and visualize the results on hexadecimal displays provided by DSCH2. The specification of this project is described in the schematic diagram of figure 5-18.

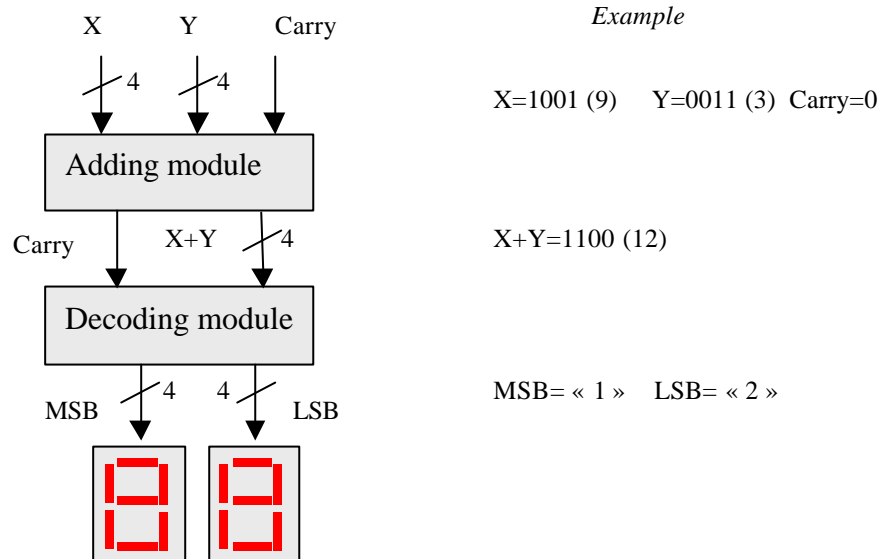


Fig.. 5-18 Flow chart of the BCD adder project

4-BIT ADDER

Four one-bit adders linked in cascade construct the 4-bit adder. You may use the adder symbol created previously, using the command "Insert -> User symbol" within DSCH2. Add two keyboard symbols and watch the result on a keyboard and a led to verify the function.

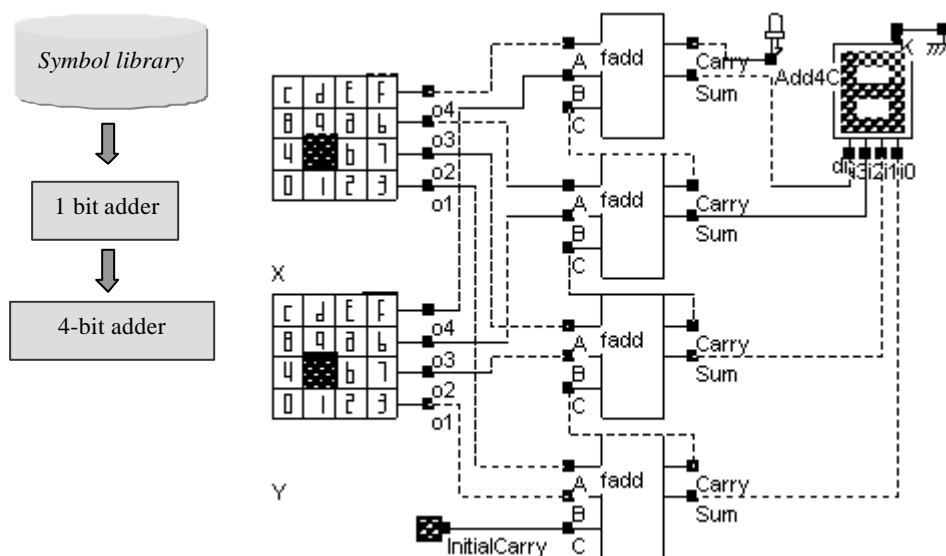
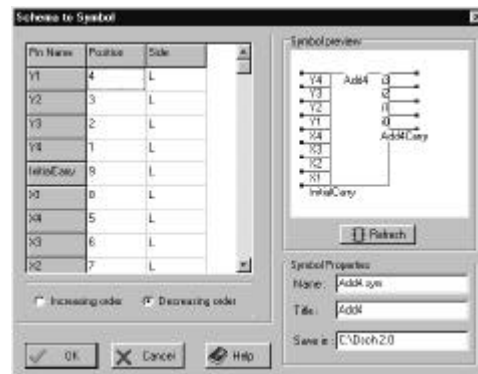


Fig. 5-19: 4-bit adder circuit (ADD4.SCH)

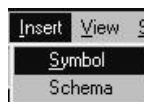
Use the following process to transfer the 4-bit adder schema into a user symbol. This symbol is tested using keyboards and digits, as shown below.



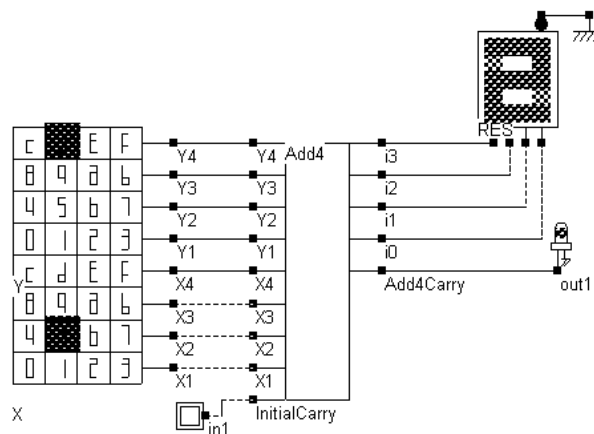
Click the icon. The schematic diagram is analyzed and a symbol is proposed in the window below. Click OK to validate the symbol. An user symbol called 'ADD4' is stored in the current directory.



Select the command 'File -> New' to restart the software.



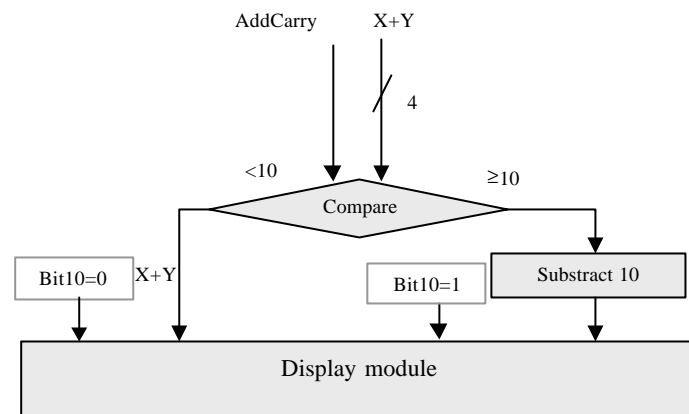
Invoke the command 'Insert ->Symbol' and choose the user symbol 'ADD4.SYM' in the list. Add Keyboards and displays to validate the behavior of this symbol.



DECODER MODULE

The objective of the decoding module is to split the binary result of the addition into two BCD codes, one representing the tenth bit ranging from 0 to 1, the other representing the unit bit ranging from 0 to 9.

The principle of the decoding circuit is shown in figure 5-20. Firstly, the $X+Y$ result is passed through a comparator. If $X+Y < 10$, the result is sent to the visualizing module. If not, the result is adjusted by subtracting 10, while the bit 10 is set to 1.



AddCarry	X+Y		≥10	Less10	Eq10
1	x	x	1	0	0
0	0	0000	0	1	0
0	1	0001	0	1	0
0	2	0010	0	1	0
0	3	0011	0	1	0
0	4	0100	0	1	0
0	5	0101	0	1	0
0	6	0110	0	1	0
0	7	0111	0	1	0
0	8	1000	0	1	0
0	9	1001	0	1	0
0	A	1010	1	0	1
0	B	1011	1	0	0
0	C	1100	1	0	0
0	D	1101	1	0	0
0	E	1110	1	0	0
0	F	1111	1	0	0

Figure 5-21: Principles and truth table of the decider module

COMPARE TO 10

The Sup10 function may be written using the following Boolean equation. The corresponding implementation is reported below. The positive logic was used for clarity, although negative logic is a better choice from delay and power consumption points of view.

$$\text{Sup10} = \text{AddCarry} + (X_3 \cdot X_2) + (X_3 \cdot X_1)$$

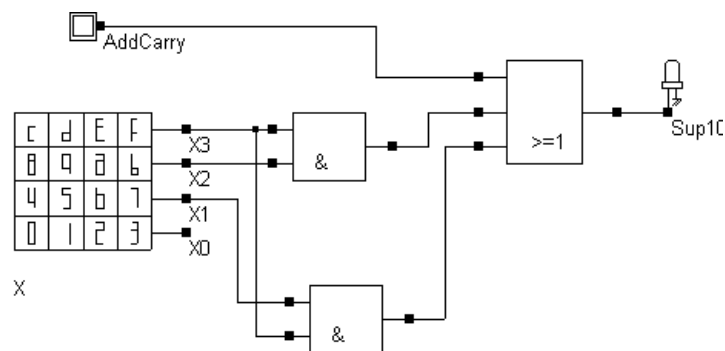


Fig. 5-22 Compare to 10 function

SUBTRACT 10

The subtraction module is enabled when the $X+Y$ result is greater or equal to 10. In that case, a subtract-by-10 operation is performed. The subtraction circuit is simply an adder with inverted input B, and an input carry set to 1. Consequently, the 4-bit subtractor can be derived from the 4-bit adder circuit, as shown below.

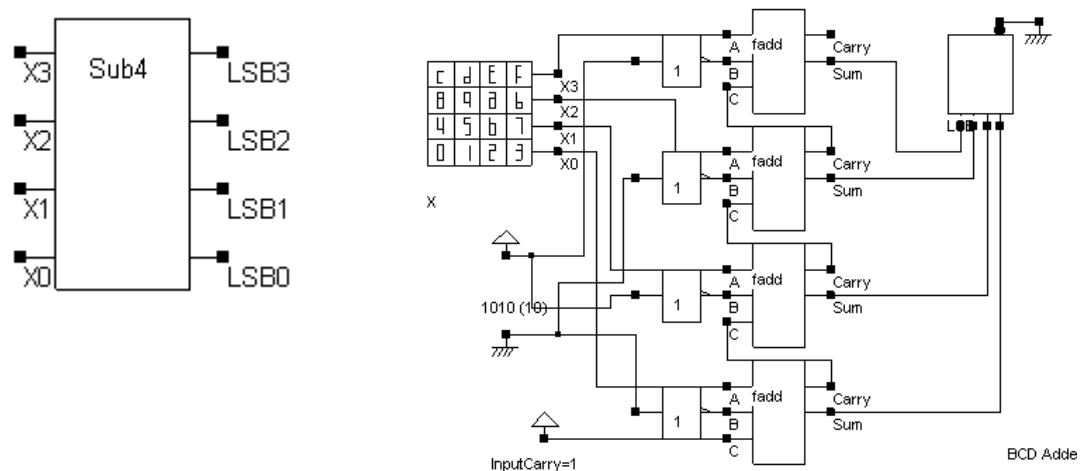


Fig. 5-23 4-bit subtractor (SUB4.SCH)

The multiplexor circuit decides whether the $X+Y$ result or the $X+Y-10$ results are sent to the display. The multiplexor is made from n-channel and p-channel MOS devices. The n-channel MOS devices multiplex the subtraction result when $Sup10$ is asserted. The p-channel MOS devices multiplex the original result to the display, when $Sup10$ is 0. The final circuit is shown in figure 5-24.

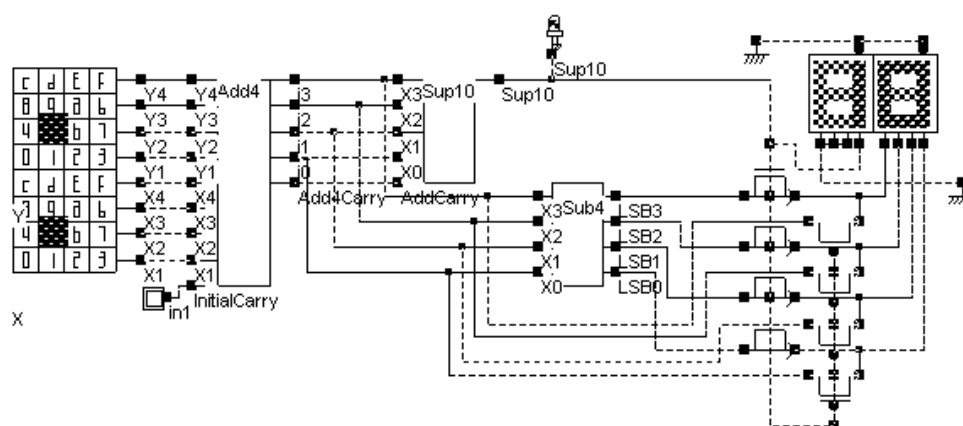


Fig. 5-24: Final circuit of the BCD adder (AdderBDC.SCH)

Conclusion

In this chapter, the design of basic arithmetic gates has been presented. The half adder and full adder have been presented. The full adder design has been conducted at layout level, with emphasis on advantages of manual design against automatic design regarding the silicon area efficiency. The comparator and multiplier circuits have also been described. A 1 bit ALU has been proposed, and a student project concerning the addition in binary-to-decimal format has been described in details.