Theory of Logic Circuits

Laboratory manual

Exercise 6

Selected arithmetic switching circuits

1. Number systems and representations

There are many different numbers systems all over the world. Most of them are weight-positional systems, where value of number's each digit depends on both value of digit and it's position in a sequence of symbols. Therefore, during labs there will be implemented circuits working with these number systems (exactly: working with numbers by a radix 2 called also binary system).

1.1. Weighted-positional number systems

In weighted-positional number systems each number is represented by sequence of digits followed by a comma or period (which depends on country). The value of a particular digit in this number depends on its position in the sequence of digits relative to a radix point. For example number *N* in weighted-positional system by base *r* can be converted to equivalent decimal number using the following polynomial function:

$$(N)_r = (d_j d_{j-1} d_{j-2} \ldots d_3 d_2 d_1 d_0 . d_{-1} d_{-2} d_{-3} \ldots d_{-k})_r$$
$$= d_j r^j + d_{j-1} r^{j-1} + d_{j-2} r^{j-2} + \ldots + d_3 r^3 + d_2 r^2 + d_1 r^1 + d_0 r^0 + d_{-1} r^{-1} + d_{-2} r^{-2} d_{-3} r^{-3} + \ldots + d_{-k} r^{-k} = (M)_{10}$$

Note: $0 \leq d_i \leq r-1$
Besides decimal, the most popular systems are also binary (base 2), octal (base 8) and hexadecimal (base 16).

1.2. Converting from decimal to other number systems

One of the methods used to convert from decimal to other number system is repeated radix division technique for integers and multiplication technique for fractions. During conversion process we must first break number into integer and fraction part. Next the integer part is repeatedly divided by new base *r* and remainders are written in reverse order (from last to first) to obtain integer part of destination number. Also fractional part is processed by repeatedly multiplying by new base *r* and every integer number obtained during each step is written in order that they appear. For example converting number $(121.375)_{10}$ into number by base 2 we obtain:

121:2 = 60 r=1
60:2=30 r=0
30:2=15 r=0
15:2=7 r=1
7:2=3 r=1
3:2=1 r=1
1:2=0 r=1

and for fraction:

0.375*2=0.75 r=0
0.75*2=1.5 r=1
0.5*2=1 r=1

so number $(119.375)_{10} = (1111001.011)_2$

1.3. Number representations

So far only positive numbers were presented. As in decimal, also in other systems we have to present both positive (represented with "+" sign) and negative (represented with "-" sign) values. This notation is called sign magnitude. Computers are generally not designed to interpret a plus or minus sign, so often the most significant bit is reserved for sign (0 means "+", 1 means "-"). During most operations computer circuits must first check sign bits of arguments and (according to their values) perform appropriate operations.
Positive and negative numbers can also be expressed by two other number representations. These are the diminished radix complement (DRC) or *(r-1)*'s complement representation, and radix complement (RC) or *r*'s complement representation.
The *(r-1)*'s complement of a number *N* is defined by the relationship, $r^i-r^{-f}-N$, where *N* is the number to be negated, *r* is the radix, *i* is the number of integer digits, and *f* is the number of fractional digits.
To obtain 1's complement of a binary number *N*, simply complement each digit of the number.
The *r*'s complement of a number *N* is defined by the relationship r^i-N, where *N* is the number to be negated, *r* is the radix, *i* is the number of integer digits, and *f* is the number of fractional digits.
To obtain 2's complement of a binary number *N*, simply first obtain 1's complement of *N* and then add 1 at the least significant bit.
Notice that positive numbers in all representations presented above are expressed in the same way.
Example (r=2):
Let $(A)_2=(11010011.01101)_2$
then we should use at least 9 positions for integer and 5 positions for fraction, so
$(A)_2=(011010011.01101)_2$
1. Using sign-magnitude representation: $(-A)_2=(\mathbf{1}11010011.01101)_2$
2. Using 1's complement representation: $(-A)_2=(\mathbf{1}00101100.10010)_2$
3. Using 2s complement representation: $(-A)_2=(\mathbf{1}00101100.10011)_2$

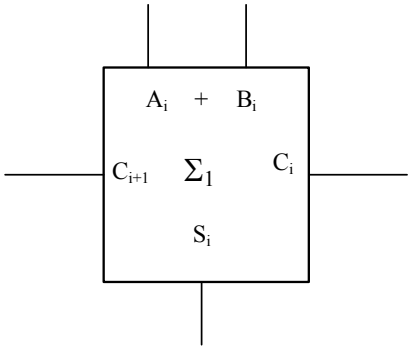
To convert negative numbers into positive, one should perform exactly the same operation as above.

2. Arithmetic operations

The operations of addition, subtraction, multiplication, and division can be carried out in exactly the same manner with binary numbers as they are with decimal numbers.
When number is in 1's or 2's complement representation, computer needs to use only Adder circuit and Complementor circuit to obtain the result of all of these operations. When we need to perform a subtraction operation, first subtrahend is converted into 1's or 2's complement and next simply added to minuend. Notice that when binary numbers to be added are in a 2's complement representation the carry occurring from the sign bit position is simply ignored, while in a 1's complement representation this carry is added back to the least significant bit of the sum.
Since multiplication can always be performed by repeated addition, and division can be repeated subtraction, the addition process can perform all four arithmetic operations.

2.1. Binary adders

Binary adder is a basic circuit performing addition operation for binary numbers A and B. This simple combinational circuit obtains two signals S_i (sum) and C_{i+1} (carry to next position) from signals A_i, B_i and C_i (carry from previous position). Fig. 1 presents the truth table and logic symbol of 1-bit adder.



C_i	A_i	B_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig.1. 1-bit adder logic symbol and truth table

There are also more complex circuits. For example in fig. 2 there is presented n-bit adder. Input signals A and B are represented as signals $A_{n-1}...A_0$ and $B_{n-1}..B_0$. Also the output sum is represented as signals $S_{n-1}..S_0$. This circuit also has carry signals: input (C_0) and output (C_n).

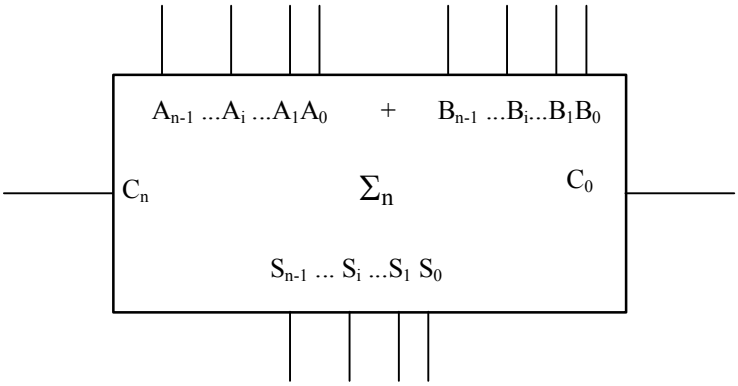
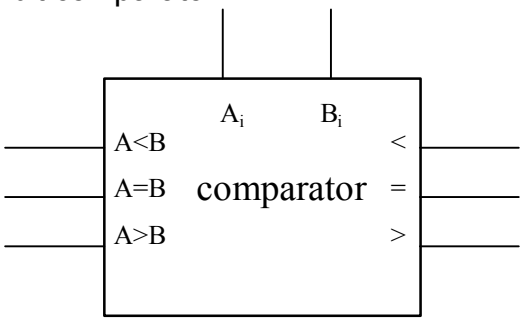


Fig.2. n-bit full adder logic symbol

2.2. Binary comparators

Another type of logic circuit used with binary numbers is a comparator. Fig. 3 presents the logic symbol and the truth table of 1-bit comparator.



<	=	>	A_i	B_i	A< B	A= B	A> B
0	0	1	0	0	0	0	1
0	0	1	0	1	1	0	0
0	0	1	1	0	0	0	1
0	0	1	1	1	0	0	1
0	1	0	0	0	0	1	0
0	1	0	0	1	1	0	0
0	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0
1	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	0	0	0	1
1	0	0	1	1	1	0	0

Fig.3. 1-bit comparator logical symbol and truth table

This circuit sets exactly one of outputs “A<B”, “A=B” and “A>B” into state 1 according to compared digits A_i and B_i and input signals “<”, “=” and “>”.
Like adders, also comparators can be realised as n-bit circuit.

2.3. Other circuits

Circuits presented above are most commonly used. There are, however, other circuits. One of them is binary subtractor. Presented in fig. 4. 1-bit subtractor realises operation A_i-B_i-C_i where C_i signal is called borrow-out bit (borrow from current position to previous).

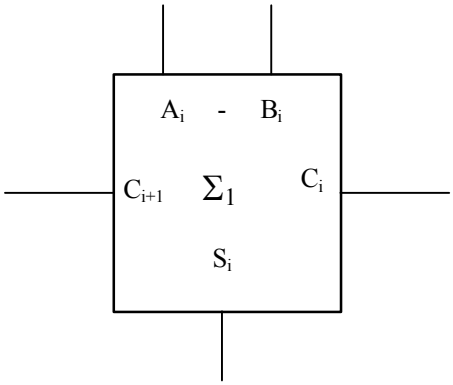


Fig.4. 1-bit subtractor logic symbol

There are also specialised circuits called arithmetic-logic unit that realise many different operations. One of them is TTL circuit 74181. This circuit realises e.g. addition, subtraction, logical multiplication. The type of operation depends on signals on function inputs $F_3..F_0$ and M.

2.4. Addition using 2's complement representation

To add two numbers written in 2's complement representation we should perform binary addition. If carry from sign digit is detected, it should be ignored.
Example:
Let $(A)_2=(010011.011)_2$ and $(B)_2=(001001.101)_2$
then $(-A)_2=(101100.101)_2$ and $(-B)_2=(110110.011)_2$

A	010011.011
+B	001001.101
A+B	**0**11101.000

Number is positive

A	010011.011
+(-B)	110110.011
A+(-B)	**0**01001.110

Number is positive.
Carry is ignored

(-A)	101100.101
+B	001001.101
(-A)+B	**1**10110.010

Number is negative.

(-A)	101100.101
+ (-B)	110110.011
(-A)+(-B)	**1**00011.000

Number is negative.
Carry is ignored

Note: Result is also in 1's complement representation (except when overflow occurred).

2.5. Addition using 1's complement representation

To add two numbers written in 1's complement representation we should also perform binary addition. If carry from sign digit is detected, it should be added on the least significant position of the result.
Example:
Let $(A)_2=(010011.011)_2$ and $(B)_2=(001001.101)_2$
then $(-A)_2=(101100.100)_2$ and $(-B)_2=(110110.010)_2$

A	010011.011	A	010011.011	(-A)	101100.100	(-A)	101100.100
+B	001001.101	+(-B)	110110.010	+B	001001.101	+(-B)	110110.010
A+B	**0**11101.000		001001.101	(-A)+B	**1**10110.001		100010.110
Number is positive		A+(-B)	**0**01001.110	Number is negative.		(-A)+(-B)	**1**00010.111
		Number is positive. Carry is added				Number is negative. Carry is added	

Note: Result is also in 1's complement representation (except when overflow occurred).

2.6. Addition using sign magnitude representation

In the case of sign magnitude representation addition is a little complicated. The addition algorithm contains several steps:

1. Check signs of both numbers.
2. If signs are equal (both are positive or negative) add magnitudes and use the sign as sign of the result.
3. If numbers have different signs, convert one (negative) of them to 1's or 2's complement representation, add both numbers using hints from sections 2.4 or 2.5 and if result is negative, convert it into sing magnitude representation.

2.7. Overflow detection

Overflow is the most important problem that can occur during addition.
Example (2's complement representation):
Let $(A)_2=(010011.011)_2$
then $(-A)_2=(101100.101)_2$

A	**0**10011.011
+A	**0**10011.011
A+A	**1**00100.110

Error: Result is negative!

(-A)	**1**01100.101
+(-A)	**1**01100.101
(-A)+(-A)	**0**11001.010

Error: Result is positive!

One of the possible solutions of detecting overflow is based on testing the carry onto and from sign digit. If both carry signals are the same, the result of addition is correct. Otherwise there was overflow error during addition operation (see examples presented above).
If overflow occurred, the number of integer digits should be increased (if it is possible).

2.8. Comparison

If we use binary comparator for numbers in sign magnitude, 1's or 2's complement representation, the result isn't correct. It is caused by sign bit, which is equal 1 for negative numbers but in binary numbers has the biggest weight. To obtain correct result the negative numbers should be complemented (in 1's and 2's complement representation), or both numbers should be tested like during addition (in sign magnitude representation).
Using sign magnitude or 1's complement representation, there is one more problem: there are two zeros: positive (e.g. 0000.0000) and negative (1000.0000 for sign magnitude or 1111.1111 for 1's complement). For these reasons comparator should use an additional combinational circuit, detecting positive and negative zeros and forcing equality one the output.

3. Tasks to be performed during laboratory

1. Obtain full 1-bit adder using NAND gates.
2. Obtain full 1-bit subtractor using NAND gates.
3. Obtain full 1-bit comparator using NAND gates.
4. Obtain full 5-bit adder using one 4-bit adder and NAND gates for numbers in:
   a) 1’s complement representation
   b) 2’s complement representation
5. Obtain full 5-bit comparator using one 4-bit comparator and NAND gates for numbers in:
   a) 1’s complement representation (positive and negative zeros are equal)
   b) 2’s complement representation

4. Instructions to follow

1. Solve all tasks before the exercise.
2. Implement the circuits specified by your supervisor (using given elements).
3. Present working circuits to your supervisor for acceptance.