Digital systems and basics of electronics - SYC

07 Introduction to digital systems

08 Minimization of Boolean functions

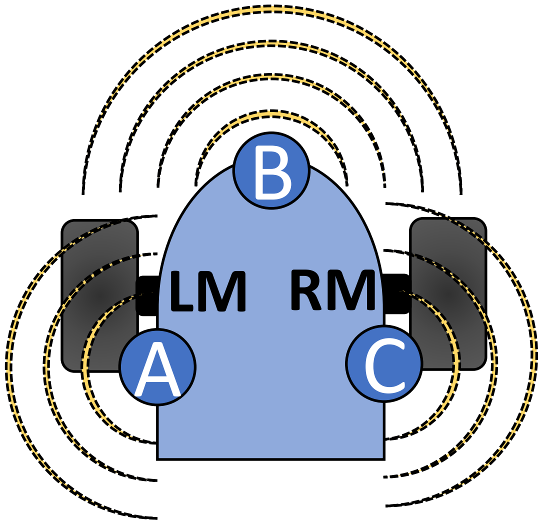
Boolean functions

Boolean functions (1-true, 0-false) are useful in designing algorithms such as the movement of a robot. We can create control logic that dictates a mobile robot's response to specific situations, like detecting obstacles, changing direction, or stopping at the appropriate moment.

3 **distance sensor** located:

A. On left side A
B. In front of B
C. On right side C

When obstacle is detected sensor gives logical "1", otherwise gives "0"

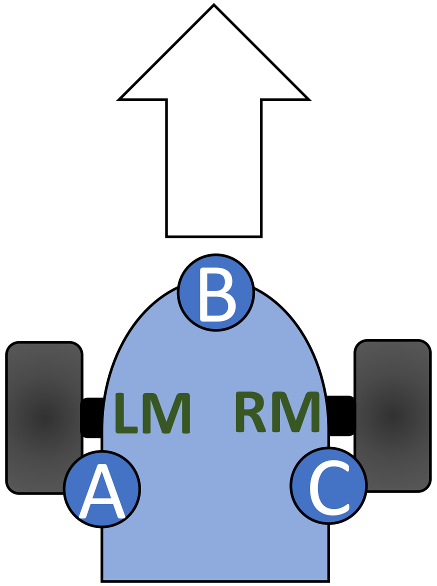


The moving platform uses differential drive: **left motor LM**, **right motor RM**. Engine is on when logical "1" control signal is applied.

Robot should move without any collision. When robot could not continue movement should stop (LM=RM=0).

1 Truth Table

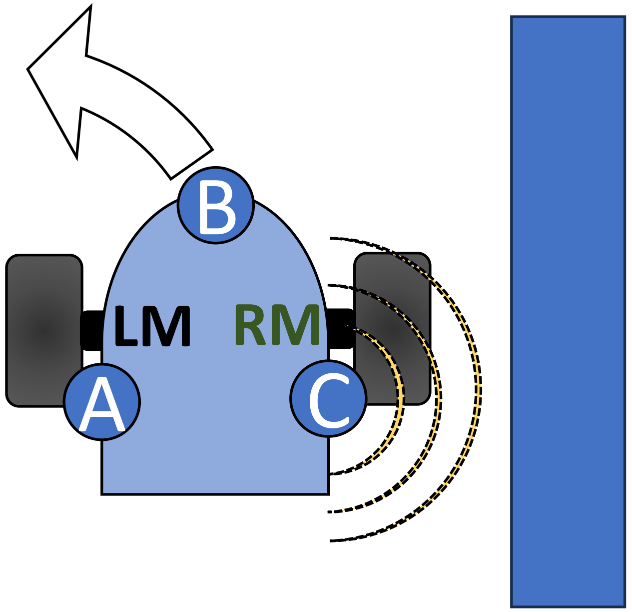
Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



Obstacle is not detected (sensor gives “0”). The robot moves forward. The left motor and right motor are set to logical true.

1 Truth Table

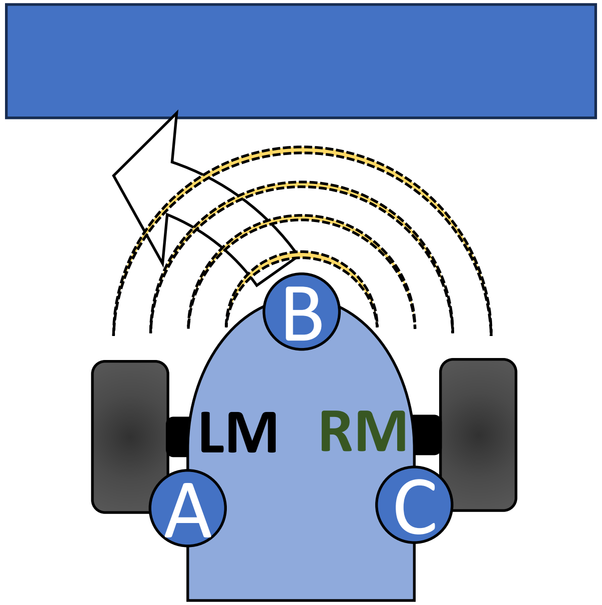
Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1	0	1
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



An obstacle is detected on the right side (C sensor gives “1”).
The robot turns left.
The right motor is set to logical true.

1 Truth Table

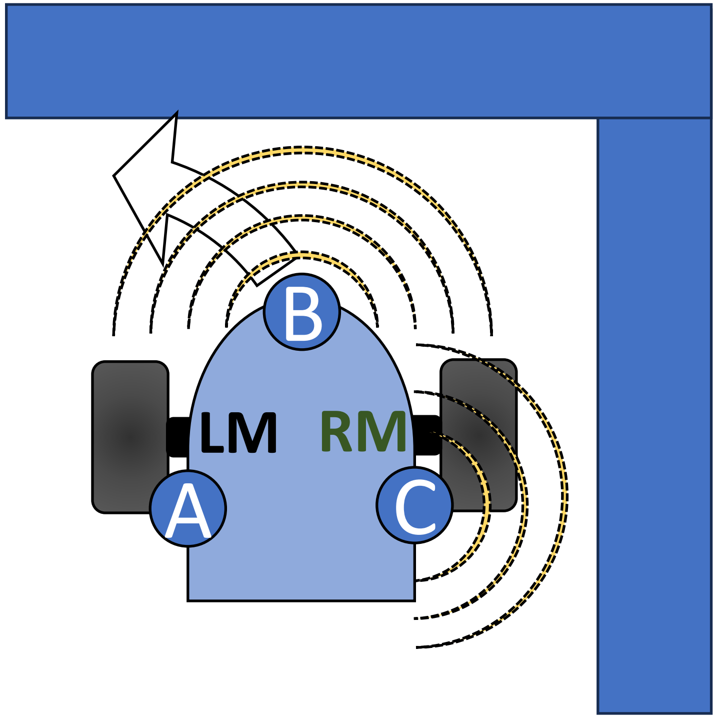
Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



An obstacle is detected in front
(B sensor gives “1”).
The robot turns left.
The right motor is set to logical true.

1 Truth Table

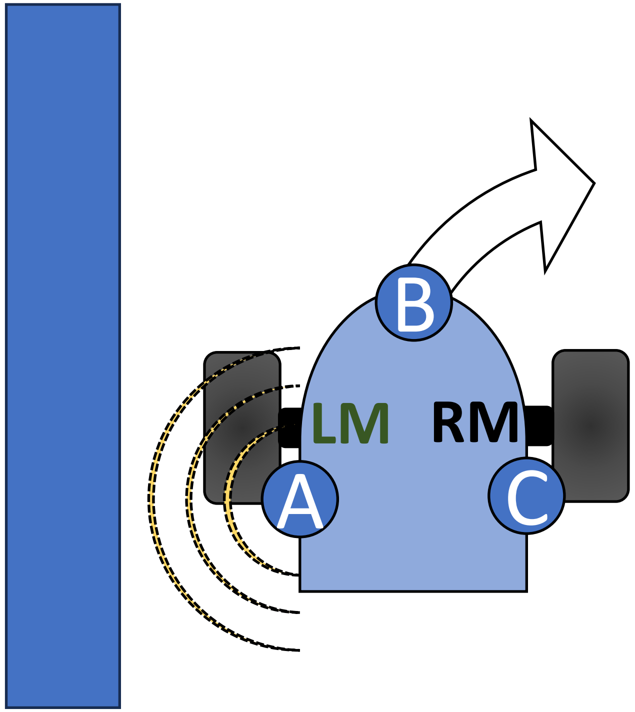
Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0		
1	0	1		
1	1	0		
1	1	1		



An obstacle is detected in front and on the right side (B and C sensors give “1”).
The robot turns left.
The right motor is set to logical true.

1 Truth Table

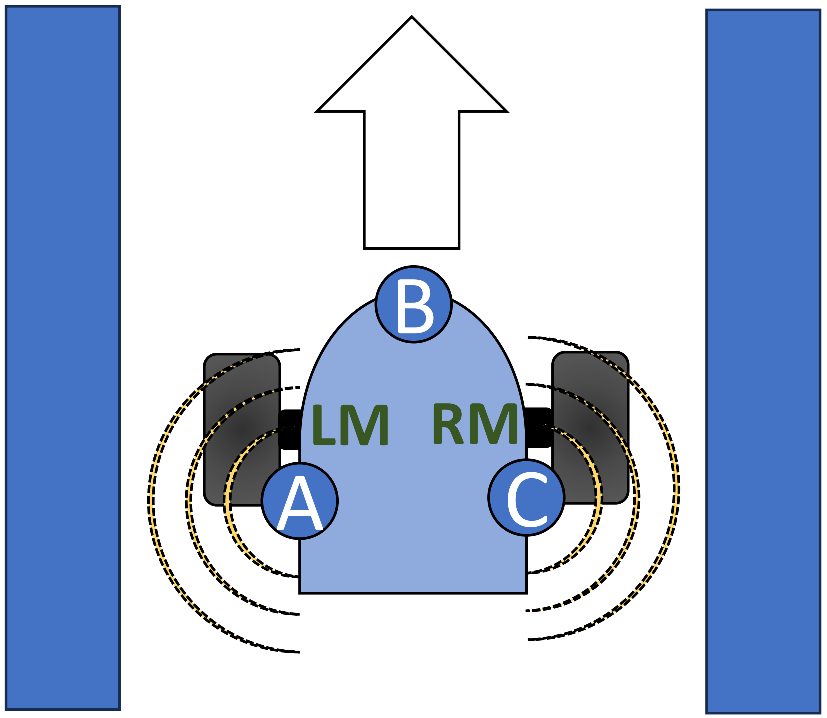
Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1		
1	1	0		
1	1	1		



An obstacle is detected on the left side
(A sensor gives “1”).
The robot turns right.
The left motor is set to logical true.

1 Truth Table

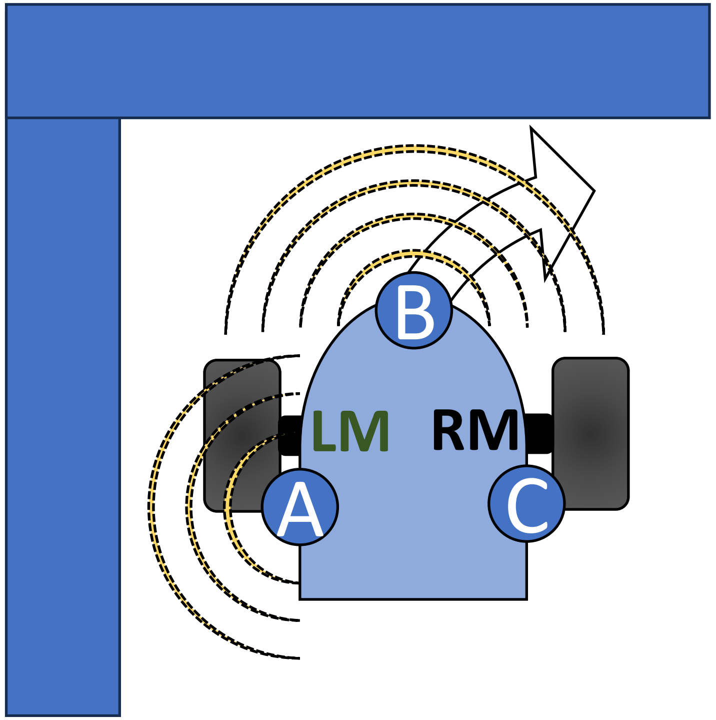
Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0		
1	1	1		



An obstacle is detected on both sides (A and C sensors give “1”).
The robot moves forward.
Both motors are set to logical true.

1 Truth Table

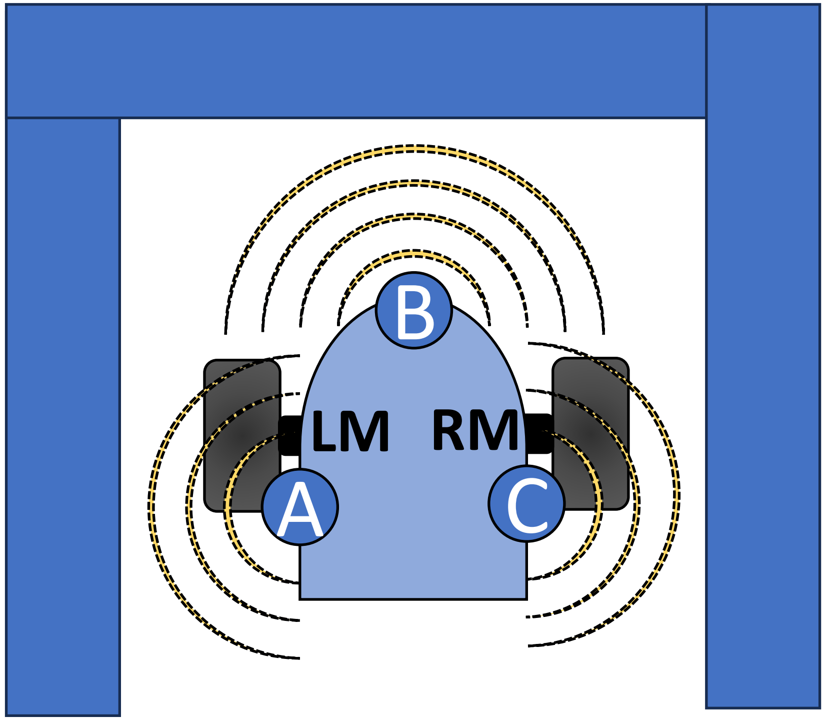
Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1		



An obstacle is detected on the left side and in front (A and B sensors give “1”).
The robot turns right.
The left motor is set to logical true.

1 Truth Table

Truth Table				
A	B	C	LM	RM
0	0	0	1	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0



An obstacle is detected on both sides and in front (A, B and C sensors give “1”). The robot stops.

2. Karnaugh map – grouping

Truth Table			
A	BC	LM	RM
0	00	1	1
0	01	0	1
0	10	0	1
0	11	0	1
1	00	1	0
1	01	1	1
1	10	1	0
1	11	0	0

LM Karnaugh map - grouping "1"					
	B C				
		00	01	11	10
A	0	1			
	1				

RM Karnaugh map - grouping "1"					
	B C				
		00	01	11	10
A	0	1			
	1				

2. Karnaugh map – grouping

Truth Table			
A	BC	LM	RM
0	00	1	1
0	01	0	1
0	10	0	1
0	11	0	1
1	00	1	0
1	01	1	1
1	10	1	0
1	11	0	0

LM Karnaugh map - grouping "1"					
	B C				
		00	01	11	10
A	0	1	0		
	1				

RM Karnaugh map - grouping "1"					
	B C				
		00	01	11	10
A	0	1	1		
	1				

2. Karnaugh map – grouping

Truth Table			
A	BC	LM	RM
0	00	1	1
0	01	0	1
0	10	0	1
0	11	0	1
1	00	1	0
1	01	1	1
1	10	1	0
1	11	0	0

LM Karnaugh map - grouping "1"					
	B C				
		00	01	11	10
A	0	1	0		0
	1				

RM Karnaugh map - grouping "1"					
	B C				
		00	01	11	10
A	0	1	1		1
	1				

2. Karnaugh map – grouping

Truth Table			
A	BC	LM	RM
0	00	1	1
0	01	0	1
0	10	0	1
0	11	0	1
1	00	1	0
1	01	1	1
1	10	1	0
1	11	0	0

LM Karnaugh map - grouping “1”					
		B C			
A		00	01	11	10
	0	1	0	0	0
	1	1	1	0	1

RM Karnaugh map - grouping “1”					
		B C			
A		00	01	11	10
	0	1	1	1	1
	1	0	1	0	0

Disjunctive Normal Form (DNF)

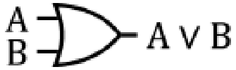
a	b	$f(a,b)$
0	0	0
0	1	$\boxed{1}$
1	0	$\boxed{1}$
1	1	0

- Disjunctive Normal Form (DNF) is a disjunction, where a clause is a conjunctive of literals. : function f is a sum of products $f = \ldots(\ldots \wedge \ldots \wedge \ldots) \vee (\ldots \wedge \ldots \wedge \ldots) \vee (\ldots \wedge \ldots \wedge \ldots)\ldots,$
- Expression in brocket (product) corresponds to one one ("1")
- In our case: $f(a,b) = \boxed{(\overline{a} \wedge b)} \vee \boxed{(a \wedge \overline{b}),}$
- Decimal description: $f(a,b) = \sum\{1,2\}$.

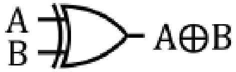
The most popular logic gates

A	NOT A
0	1
1	0

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

3. Boolean function f(a,b,c) from Karnaugh map:

LM Karnaugh map - grouping “1”					
		B C			
A		**00**	**01**	**11**	**10**
	0	1	0	0	0
	1	1	1	0	1

(not a and not b and not c) or (a and not b and not c) = (not a or a) and not b and not c = 1 and not b and not c = **not b and not c**

RM Karnaugh map - grouping “1”					
		B C			
A		**00**	**01**	**11**	**10**
	0	1	1	1	1
	1	0	1	0	0

3. Boolean function f(a,b,c) from Karnaugh map:

LM Karnaugh map - grouping “1”					
		B C			
A		**00**	**01**	**11**	**10**
	0	1	0	0	0
	1	1	1	0	1

RM Karnaugh map - grouping “1”					
		B C			
A		**00**	**01**	**11**	**10**
	0	1	1	1	1
	1	0	1	0	0

(not a and not b and not c) or (a and not b and not c) = (not a or a) and not b and not c = 1 and not b and not c = **not b and not c**

(a and not b and not c) or (a and b and not c)=
a and (not b or b) and not c = a and 1 and not c=
= **a and not c**

a and not b

3. Boolean function f(a,b,c) from Karnaugh map:

LM Karnaugh map - grouping “1”					
		B C			
A		**00**	**01**	**11**	**10**
	0	1	0	0	0
	1	1	1	0	1

RM Karnaugh map - grouping “1”					
		B C			
A		**00**	**01**	**11**	**10**
	0	1	1	1	1
	1	0	1	0	0

not b and c

not b and not c

a and not c

a and not b

(not a and not b and not c) or (not a and not b and c) or
(not a and b and c) or (not a and b and not c) =
[not a and not b and (not c or c)] or [not a and b and (c or not c)] =
[not a and not b and 1] or [not a and b and 1] =
[not a and not b] or [not a and b] =
not a and (not b or b) = not a and 1 = **not a**

4. Using Logisim, create LM and RM logic circuits (use OR, AND, NOT gates) and check it.

LM f(a,b,c) =

= (not b and not c) or (a and not c) or (a and not b)

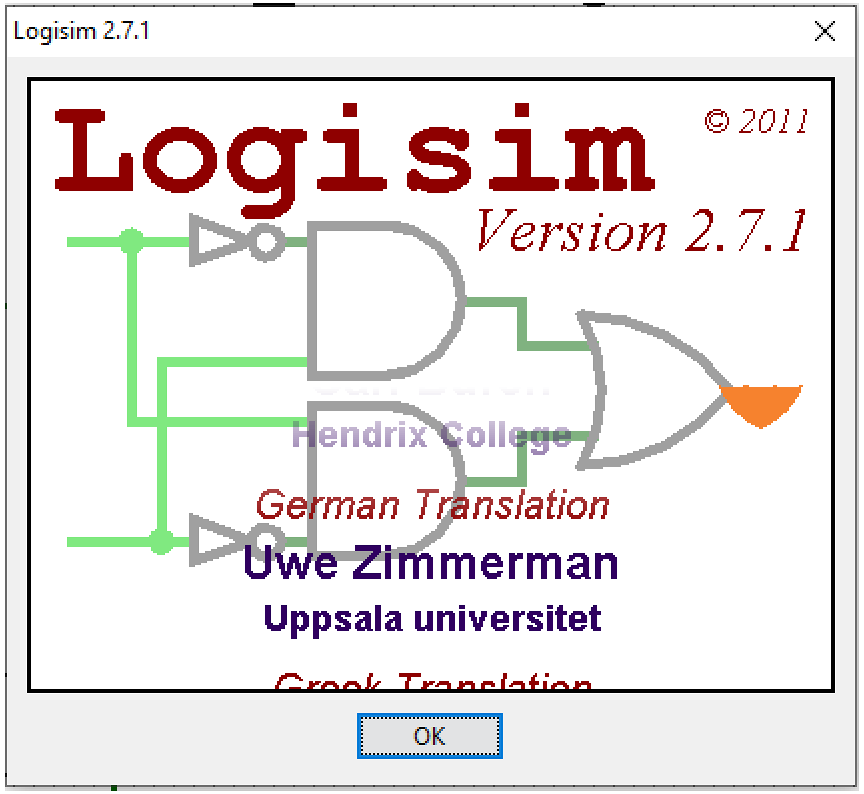
RM f(a,b,c) =

= (not a) or (not b and c)

Start logisim

Icon

Version

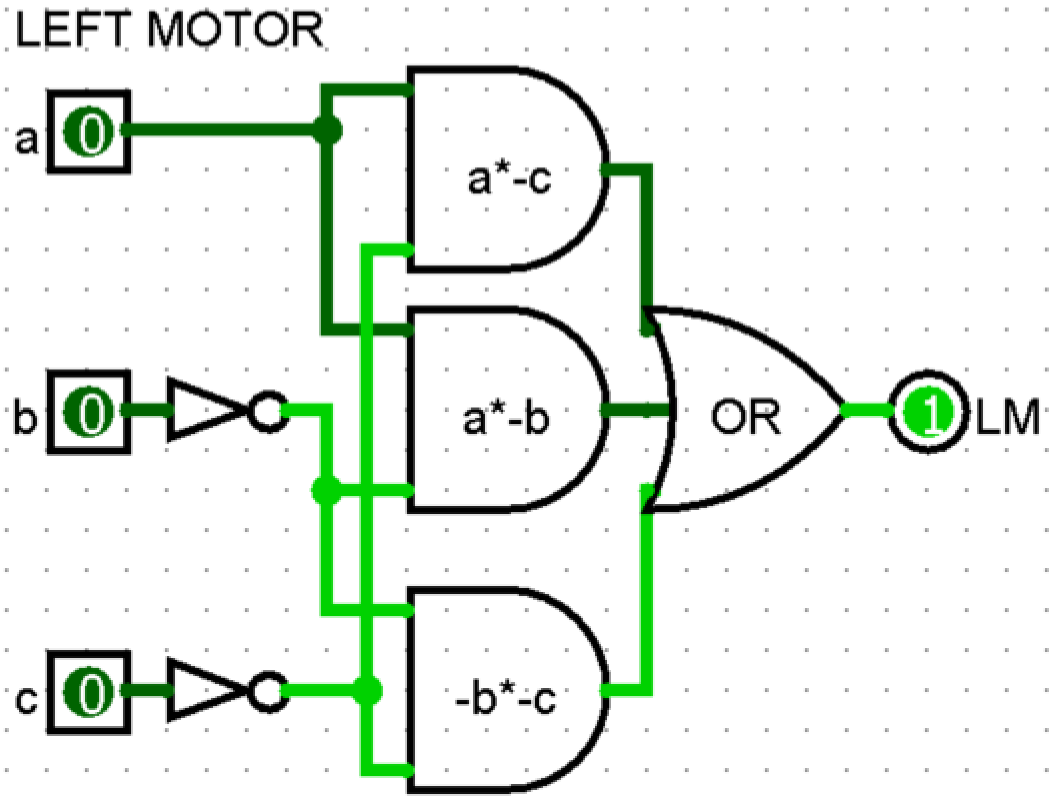


Open : mmajew/SYC/07 and 08/**07 and 08 lab mobile robot.circ**

4. Using Logisim, create LM and RM logic circuits (use OR, AND, NOT gates) and check it.

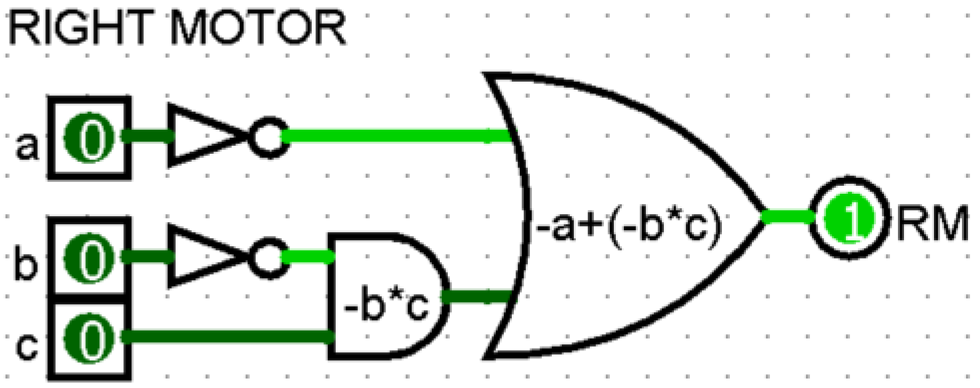
LM f(a,b,c) =

= (not b and not c) or (a and not c) or (a and not b)

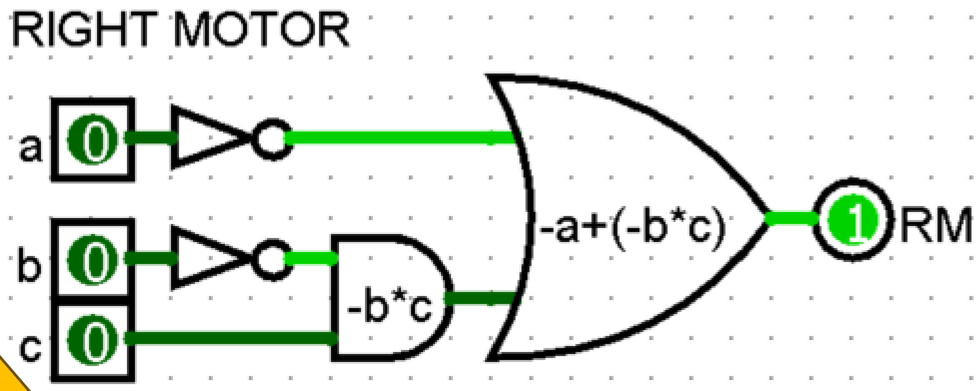


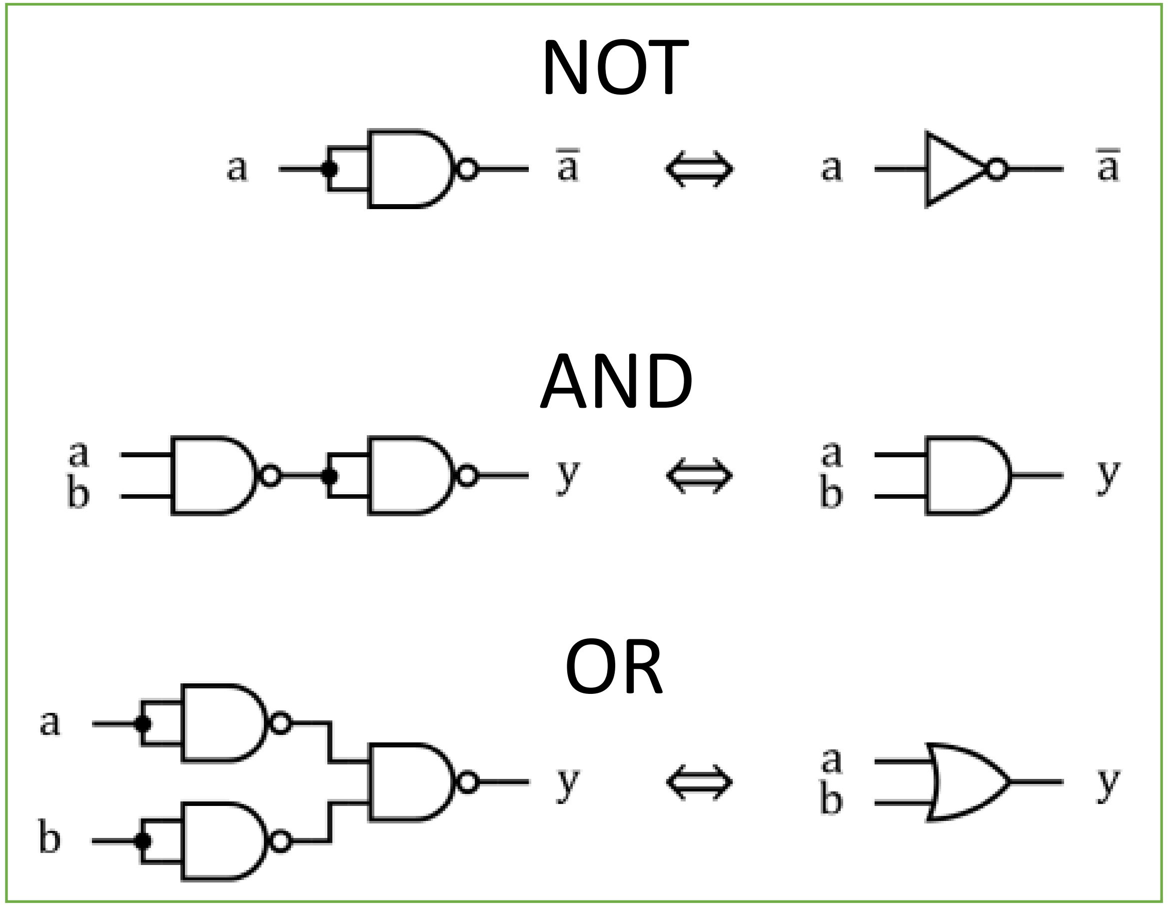
RM f(a,b,c) =

= (not a) or (not b and c)



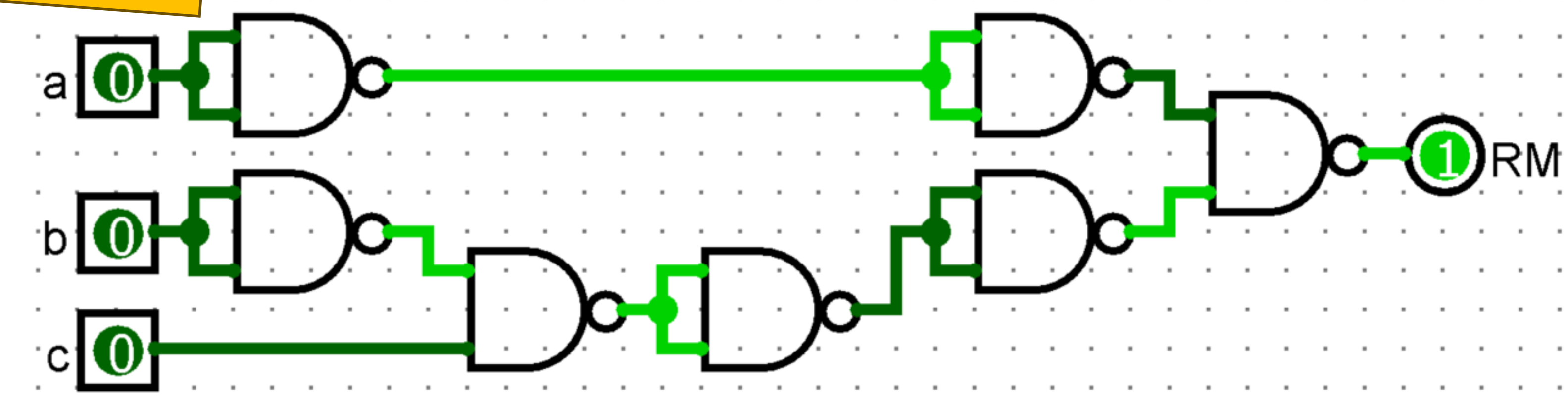
5. In Logisim create RM logic circuits. Replace OR, AND, NOT by NAND and check it.





let's do it

NAND substitution, without simplifying/minimizing the function



it's quite complicated!

6. Simplify boolean function f(a,b,c) using NOT, NAND gates only:

…with simplifying/minimizing the function

LM f(a,b,c) =

= (not b and not c) or (a and not c) or (a and not b) =

RM f(a,b,c) =

= (not a) or (not b and c) =

6. Simplify boolean function f(a,b,c) using NOT, NAND gates only:

…with <u>simplifying</u>/minimizing the function

Double negation

Q = not[not(Q)]

LM f(a,b,c) =

= (not b and not c) or (a and not c) or (a and not b) =

= NOT{ NOT[(not b and not c) or (a and not c) or (a and not b)]}

RM f(a,b,c) =

= (not a) or (not b and c) =

= NOT{ NOT[(not a) or (not b and c)]}

6. Simplify boolean function f(a,b,c) using NOT, NAND gates only:

…with <u>simplifying</u>/minimizing the function

Double negation

Q = not[not(Q)]

De Morgan's law

not(P or Q) = (not P) and (not Q)

not(P or Q or R) = (not P) and (not Q) and (not R)

LM f(a,b,c) =

= (not b and not c) or (a and not c) or (a and not b) =

= NOT{ NOT[(not b and not c) or (a and not c) or (a and not b)]} =

= NOT{ NOT(not b and not c) and NOT(a and not c) and NOT(a and not b)}

RM f(a,b,c) =

= (not a) or (not b and c) =

= NOT{ NOT[(not a) or (not b and c)]} =

= NOT{ NOT(not a) and NOT(not b and c) }

6. Simplify boolean function f(a,b,c) using NOT, NAND gates only:

…with <u>simplifying</u>/minimizing the function

Double negation

$Q = not[\ not(\ Q\ )\ ]$

De Morgan's law

not(P or Q) = (not P) and (not Q)

not(P or Q or R) = (not P) and (not Q) and (not R)

LM f(a,b,c) =

= (not b and not c) or (a and not c) or (a and not b) =

= NOT{ NOT[(not b and not c) or (a and not c) or (a and not b)]} =

= NOT{ NOT(not b and not c) and NOT(a and not c) and NOT(a and not b)} =

= NOT{ (not b Nand not c) and (a Nand not c) and (a Nand not b)}

RM f(a,b,c) =

= (not a) or (not b and c) =

= NOT{ NOT[(not a) or (not b and c)]} =

= NOT{ NOT(not a) and NOT(not b and c) } =

= NOT{ a and (not b Nand c) }

6. Simplify boolean function f(a,b,c) using NOT, NAND gates only:

…with simplifying/minimizing the function

Double negation

Q = not[not(Q)]

De Morgan's law

not(P or Q) = (not P) and (not Q)

not(P or Q or R) = (not P) and (not Q) and (not R)

LM f(a,b,c) =

= (not b and not c) or (a and not c) or (a and not b) =

= NOT{ NOT[(not b and not c) or (a and not c) or (a and not b)]} =

= NOT{ NOT(not b and not c) and NOT(a and not c) and NOT(a and not b)} =

= NOT{ (not b Nand not c) and (a Nand not c) and (a Nand not b)} =

= (not b Nand not c) Nand (a Nand not c) Nand (a Nand not b)

RM f(a,b,c) =

= (not a) or (not b and c) =

= NOT{ NOT[(not a) or (not b and c)]} =

= NOT{ NOT(not a) and NOT(not b and c) } =

= NOT{ a and (not b Nand c) } =

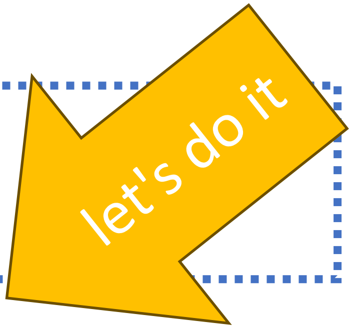
= a Nand (not b Nand c)

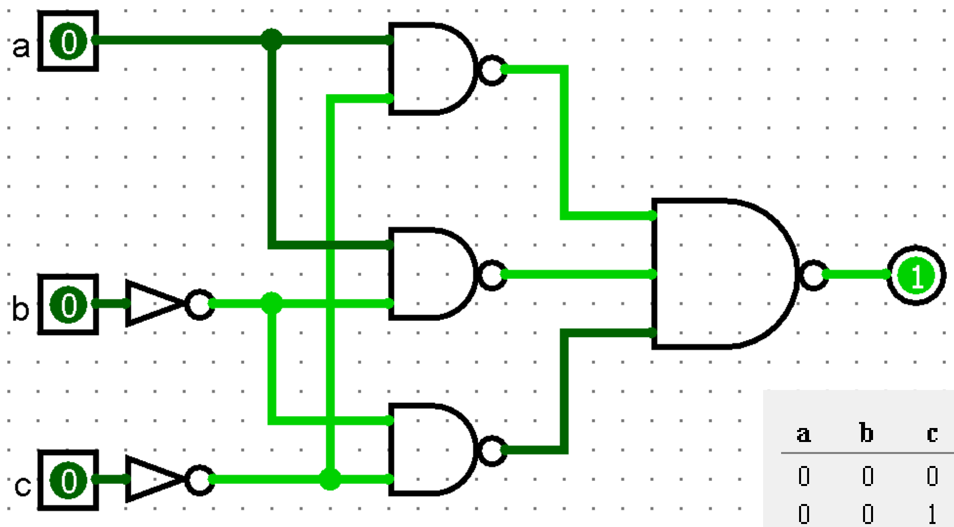
7. Using Logisim, create above LM and RM logic circuits (NAND, NOT gates) and check it.

LM f(a,b,c) = (not b Nand not c) Nand (a Nand not c)

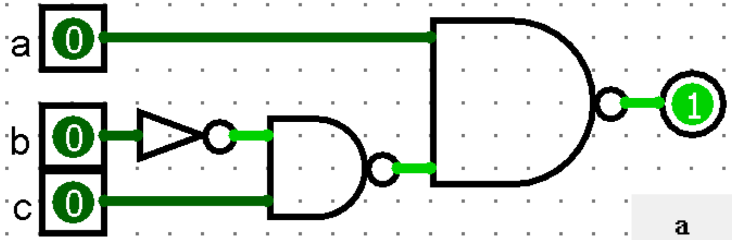
Nand (a Nand not b)

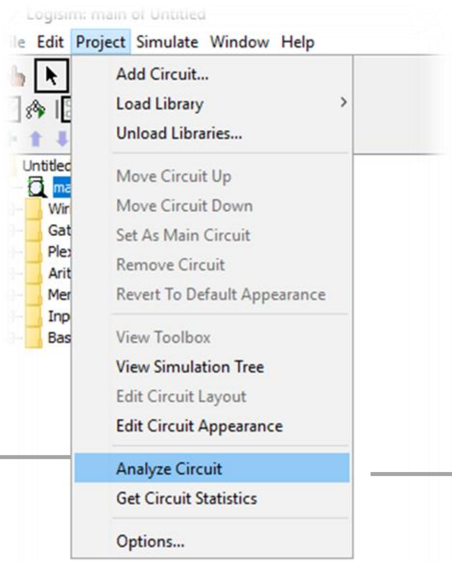
RM f(a,b,c) = a Nand (not b Nand c)





a	b	c	x
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





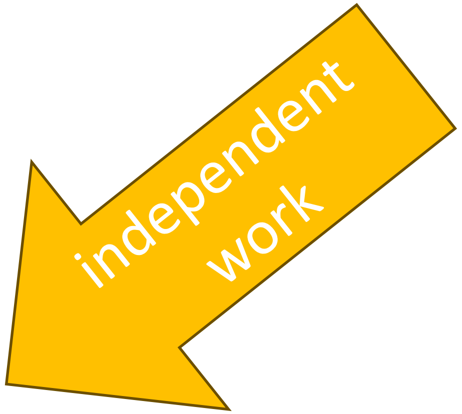
a	b	c	x
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0				
	1				



1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ______________________________

____

____

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1.A Truth tab			
a	b	c	f( )
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0				
	1				

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ____________________

____

____

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, **lab 08**			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"

		bc			
		00	01	11	10
a	0				
	1				

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ____________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table

a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"

a \ bc	00	01	11	10
0	1			
1				

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ____________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"

a \ bc	00	01	11	10
0	1	1		
1				

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ______________________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0	1	1		0
	1				

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ______________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0	1	1	1	0
	1				

1

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ______________________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0	1	1	1	0
	1	0			

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ____________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0	1	1	1	0
	1	0	1		

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ______________________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0	1	1	1	0
	1	0	1		0

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ____________________

____

____

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, **lab 08**			
Student number		Number of points	

1. Solve 1.B to 1.F tasks

1.A Truth table

a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"

a \ bc	00	01	11	10
0	1	1	1	0
1	0	1	0	0

1

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = ______________________

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1.A Truth table

a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"

a \ bc	00	01	11	10
0	1	1	1	0
1	0	1	0	0

1

(not a and not b and <u>not c</u>) or (not a and not b and <u>c</u>)

not a and not b and <u>**(not c or c)**</u> =
not a and not b and <u>**1**</u> =
not a and not b

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = (not a and not b) or

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table

a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"

		bc			
		00	01	11	10
a	0	1	1	1	0
	1	0	1	0	0

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = (not a and not b) or (not a and c) or (not b and c)

1.B Karnaugh map - grouping "1"

a \ bc	00	01	11	10
0	1	0	0	1
1	1	0	1	0

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = (not a and not c) or (not b and not c) or (a and b and c)

1.B Karnaugh map - grouping "1"

a \ bc	00	01	11	10
0	0	1	1	0
1	1	1	1	0

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = (c) or (a and not b);

[not a and not b and c] or [not a and b and c] or [a and **not b** and c] or [a and **b** and c] =

[not a and and c and (not b or b)] or [a and and c and **(not b or b)**] =

[not a and c] or [a and c] =

c and (not a and a) =

c

Polish Japanese Academy of Information Technology			
Digital systems and basics of electronics, ***lab 08***			
Student number		*Number of points*	

1. Solve 1.B to 1.F tasks

1

1.A Truth table			
a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1.B Karnaugh map - grouping "1"					
		bc			
		00	01	11	10
a	0	1	1	1	0
	1	0	1	0	0

1.C Boolean function f(a,b,c) from Karnaugh map:

f(a,b,c) = (not a and not b) or (not a and c) or (not b and c)

1.D Boolean function f(a,b,c) using NAND gates only:

(not a and not b) or (not a and c) or (not b and c)=

CAT or DOG or HOUSE

1.D Boolean function f(a,b,c) using NAND gates only:

(not a and not b) or (not a and c) or (not b and c)=

CAT or DOG or HOUSE =

double negation → Not{ Not[CAT or DOG or HOUSE] }

Q = not[not(Q)]

1.D Boolean function f(a,b,c) using NAND gates only:

(not a and not b) or (not a and c) or (not b and c)=

CAT or DOG or HOUSE =

double negation

Q = not[not(Q)]

Not{ Not[CAT or DOG or HOUSE] }=

De Morgan's law

not(P or Q) = (not P) and (not Q)

not(P or Q or R) =

(not P) and (not Q) and (not R)

Not{ (not CAT) and (not DOG) and (not HOUSE) }

1.D Boolean function f(a,b,c) using NAND gates only:

(not a and not b) or (not a and c) or (not b and c)=

CAT or DOG or HOUSE =

double negation
Q = not[not(Q)]

Not{ Not[CAT or DOG or HOUSE] }=

De Morgan's law
not(P or Q) = (not P) and (not Q)

Not{ (not CAT) and (not DOG) and (not HOUSE) }=

Not{ (not CAT) and (not DOG) and (not HOUSE) }=

(not CAT) **Nand** (not DOG) **Nand** (not HOUSE)

1.D Boolean function f(a,b,c) using NAND gates only:

(not a and not b) or (not a and c) or (not b and c)=

CAT or DOG or HOUSE =

double negation

Q = not[not(Q)]

Not{ Not[CAT or DOG or HOUSE] }=

De Morgan’s law

not(P or Q) = (not P) and (not Q)

Not{ (not CAT) and (not DOG) and (not HOUSE) }=

Not{ (not CAT) and (not DOG) and (not HOUSE) }=

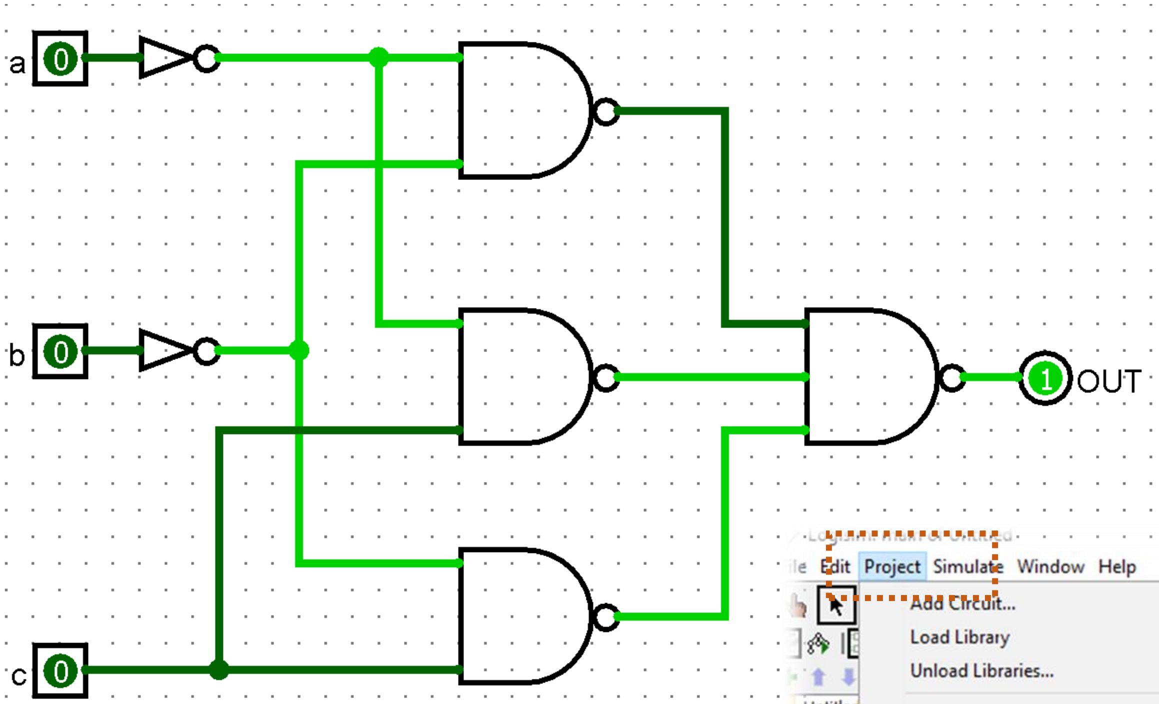
(not CAT) **Nand** (not DOG) **Nand** (not HOUSE) =

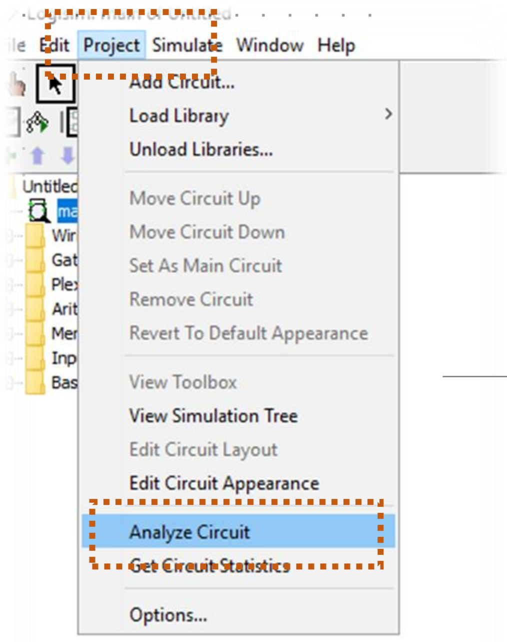
(not (not a and not b)) **Nand** (not (not a and c)) **Nand** (not (not b and c)) =

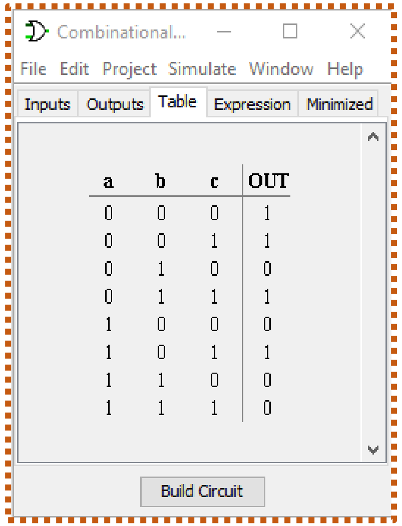
(not a **Nand** not b) **Nand** (not a **Nand** c) **Nand** (not b **Nand** c)

Boolean function using Not and Nand gates

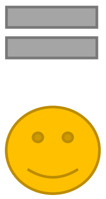
1.E Using Logisim, create a logic circuit and check it.







a	b	c	OUT
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



1.A Truth table

a	b	c	f(a,b,c)
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0