

Title: “DON’T CARE” functions in equivalent logic expressions

Author: Peter Lablans, Ternarylogic LLC.

Date: July 7, 2005

Equivalent logic expressions

Equivalent logic expressions for the purpose of this article are logic expressions with an identical number of logic functions and an identical number of input variables which will generate identical output states for identical values of the input states.

The purpose of this article is to describe how different logic functions can create equivalent logic expressions. One of the consequences of creating equivalent expressions is the occurrence of “DON’T CARE” functions. Rather than try to eliminate these functions they will be investigated on why and how they occur.

The full binary adder

The composite expression that will be used is the one for realizing a full binary adder. The binary sum of two single digits ‘a’ and ‘b’ is formed by the modulo-2 residue ‘res1’ of ‘a’ and ‘b’ and the modulo-2 carry ‘car1’ of ‘a’ and ‘b’.

$$\begin{aligned} \text{res1} &\rightarrow (a \neq b) \\ \text{car1} &\rightarrow (a \wedge b) \end{aligned}$$

The sum is formed by the two digits [car1 res1].

Determining the full adder sum of multi-digit number is a process that comprises a repeated execution of the Residue and the Carry steps. In general the execution is considered to be a serial process, as a next step has to wait for a previous step in order to be executed. It does not have to be that way, however that is beyond the scope of this article. The first two steps of the full addition of two 4 digit binary numbers [a b c d] and [e f g h] is described in the following table.

number1		a	b	c	d
number2		e	f	g	h
res1		(a ≠ e)	(b ≠ f)	(c ≠ g)	(d ≠ h)
car1	(a ∧ e)	(b ∧ f)	(c ∧ g)	(d ∧ h)	-

The intermediate results of the execution are the individual bits provided by ‘res1’ and ‘car1’.

The next results res2 and car2 will be created by repeating the previous steps on the generated results. In fact we may consider the numbers ‘number1’ and ‘number2’ to have been replaced by the two numbers ‘res1’ and ‘car1’.

number1		a	b	c	d
number2		e	f	g	h
res1		$(a \neq e)$	$(b \neq f)$	$(c \neq g)$	$(d \neq h)$
car1	$(a \wedge e)$	$(b \wedge f)$	$(c \wedge g)$	$(d \wedge h)$	-
res2	$(b \neq f) \neq (c \wedge g)$	$(c \neq g) \neq (d \wedge h)$	
car2	$(c \neq g) \wedge (d \wedge h)$		-

In the second cycle the generated intermediate results are 'res2' and 'car2'. As is known from the full adder, the sum will be created going from right to left as the carry is being resolved and will disappear as an influence on the result.

We may consider the bits in row 'res2' and 'car2' in the column under b and f to be the canonical form of the second cycle residue and carry bits.

$$\text{res2} \rightarrow (b \neq f) \neq (c \wedge g)$$

$$\text{car2} \rightarrow (c \neq g) \wedge (d \wedge h)$$

One can name:

A equivalent to: $(b \neq f)$

B equivalent to: $(c \wedge g)$

C equivalent to: $(c \neq g)$

D equivalent to: $(d \wedge h)$

In that case 'res2' and 'car2' can be rewritten as:

$$\text{res2} \rightarrow A \neq B$$

$$\text{car2} \rightarrow C \wedge D$$

It should be clear then that one can divide the complete full adder into these cycles and make the results of these cycles the inputs to self-similar meta-cycles.

Finding equivalent expressions

It is not possible to find equivalent expressions to the first cycle, which we may call 'primitive'. It is possible to find equivalent expressions for the second cycle.

First consider: $\text{res2} \rightarrow (a \neq b) \neq (c \wedge d)$.

There are 3 equivalent expressions for this one:

$$\text{res2} \rightarrow (a \neq b) = (c \text{ NAND } d)$$

$$\text{res2} \rightarrow (a = b) \neq (c \text{ NAND } d)$$

$$\text{res2} \rightarrow (a = b) = (c \wedge d)$$

As more equivalent expressions will be shown it will help to change the notation of the expressions.

The generic form of the logic expression is:

$$\text{res2} \rightarrow (a \text{ r1 } b) \text{ r2 } (c \text{ r3 } d)$$

The terms r1, r2 and r3 signify binary logic functions. The functions are represented by a number that will relate to the respective truth tables. The following table shows the 14 non-trivial truth tables (all 0 and all 1 truth tables are not included).

1	0	1	2	0	1	3	0	1	4	0	1	5	0	1	6	0	1	7	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1
1	0	1	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0	1	1	1

8	0	1	9	0	1	10	0	1	11	0	1	12	0	1	13	0	1	14	0	1
0	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0

It is assumed that the structure of the expression remains the same. Priority of execution is determined by parentheses. The order of the variable determines the order of execution in case of non-commutative functions.

The equivalent expressions for 'res2' and 'car2' are:

expression	r1	r2	r3	
res2	9	9	1	
res2	9	6	14	
res2	6	6	1	original
res2	6	9	14	
car2	9	4	1	
car2	9	8	14	
car2	6	1	1	original
car2	6	2	14	

The expression $car2 \rightarrow (a \ 9 \ b) \ 8 \ (c \ 14 \ d)$ does not contain any of the primitive functions but will generate the same result as $car2 \rightarrow (a \ 6 \ b) \ 1 \ (c \ 1 \ d)$ or $car2 \rightarrow (a \neq b) \wedge (c \wedge d)$.

For clarity it should be emphasized that the above table is the solution for the equation $result \rightarrow (a \ r1 \ b) \ r2 \ (c \ r3 \ d)$ wherein result, a, b, c and d are the constants and r1, r2 and r3 are the variables. Or in other words: the functions are the variables in the equations and the inputs and output are the 'known entities'.

Extending the cycle

In general the expressions for the full ripple adder for n digit numbers require (n+1) expressions to create all possible sums. Because of the self-similarity of the expressions, the series of expressions can be divided into sub-cycles. The previous example showed how the results, which will be inputs to following expressions can be determined from expressions comprised of 3 functions with 4 single digit (bits) inputs.

The expressions can be extended to include a following step generating ‘res3’ and ‘car3’. This is shown in the following table.

In this Six-Input-Single-Output (SISO) approach the previously described expression with one additional cycle. The outputs ‘res3’ and ‘car3’ are calculated from 6 different input bits and generate one output bit (either res3 or car3), using potentially 7 functions.

number 1	a	b	c	d
number 2	e	f	g	h
res3	$\{(a \neq e) \neq (b \wedge f)\} \neq \{(b \neq f) \wedge (c \wedge g)\}$			
car3	$\{(b \neq f) \neq (c \wedge g)\} \wedge \{(c \neq g) \wedge (d \wedge h)\}$			

The expressions: $\text{res3} \rightarrow \{(a \neq e) \neq (b \wedge f)\} \neq \{(b \neq f) \wedge (c \wedge g)\}$ and $\text{car3} \rightarrow \{(b \neq f) \neq (c \wedge g)\} \wedge \{(c \neq g) \wedge (d \wedge h)\}$ may be considered the canonical expressions for the SISO approach.

The purpose of this article is to deal with ‘equivalent’ expressions, not to create a complete n-bits word full adder. No further attention will be given to the ‘real’ full adder.

In order to create neutral ‘canonical’ expressions the SISO expressions will be re-written as:

$\text{res3} \rightarrow \{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} \rho_3 \{(c \rho_4 d) \gamma_2 (e \gamma_3 f)\}$ and $\text{car3} \rightarrow \{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} \gamma_2 \{(c \rho_3 d) \gamma_2 (e \gamma_4 f)\}$. The previous expressions are a specific solution to the canonical equation, with the added comment that the functions (and not the inputs or output) are the variables.

First the results for the res3 expression will be determined. The equation:

$\text{res3} \rightarrow \{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} \rho_3 \{(c \rho_4 d) \gamma_2 (e \gamma_3 f)\}$ will be solved by applying a computer program, going through all possible solutions. It is not viable to run all combination tests in a workable time span. However based on earlier assumptions tests may be limited to functions wherein ρ_3 is either “=” or “≠”.

At least 128 equivalent expressions can be found. The possible equivalents are shown in the following table.

res3 → {(a ρ ₁ b) ρ ₂ (c γ ₁ d)} ρ ₃ {(c ρ ₄ d) γ ₂ (e γ ₃ f)}						
ρ ₁	ρ ₂	γ ₁	ρ ₃	ρ ₄	γ ₂	γ ₃
6	6	1	6	6	1	1
9	6	14	6	6	1	1
6	9	14	6	6	1	1
9	9	1	6	6	1	1
6	6	7	6	6	1	14
9	6	8	6	6	1	14
6	9	8	6	6	1	14
9	9	7	6	6	1	14
6	6	1	6	9	8	14
9	6	14	6	9	8	14
6	9	14	6	9	8	14
6	6	8	6	6	11	1
6	6	14	9	6	1	1
9	6	1	9	6	1	1
9	9	14	9	6	1	1
9	6	7	9	6	1	14
9	6	7	9	9	2	14
9	9	8	9	6	4	1
9	6	8	9	6	11	1
9	6	8	9	9	13	14
9	9	7	9	6	14	14

The first solution is of course the original equation with the primitive functions 6 and 1. The equivalent expressions apply non-commutative functions like 2, 4, 11, and 13.

At least 2500 equivalent expressions can be found for 'car3'. Some examples:

car3 → {(a ρ ₁ b) ρ ₂ (c γ ₁ d)} γ ₂ {(c ρ ₃ d) γ ₃ (e γ ₄ f)}						
ρ ₁	ρ ₂	γ ₁	γ ₂	ρ ₃	γ ₃	γ ₄
6	6	1	1	6	1	6
9	5	0	2	9	2	1
9	5	1	2	9	2	1
9	5	2	2	9	2	1
9	5	3	2	9	2	1
9	5	4	2	9	2	1
9	5	5	2	9	2	1
9	5	6	2	9	2	1
9	5	7	2	9	2	1
9	5	8	2	9	2	1
9	5	9	2	9	2	1
6	14	6	2	0	3	1
9	7	9	2	1	3	1
6	11	9	2	2	3	1
6	14	6	2	15	14	14
6	14	6	2	6	14	14
6	10	0	8	9	13	1
6	10	1	8	9	13	1
6	10	2	8	9	13	1
6	10	3	8	9	13	1
6	10	4	8	9	13	1
6	10	5	8	9	13	1
6	10	6	8	9	13	1
6	10	7	8	9	13	1
6	10	8	8	9	13	1
6	10	9	8	9	13	1
6	10	10	8	9	13	1
6	10	11	8	9	13	1
6	10	12	8	9	13	1
6	10	13	8	9	13	1
6	10	14	8	9	13	1
6	10	15	8	9	13	1
9	9	9	4	9	7	14

The first solution in the table is the original solution. The following 10 solutions are of interest because the function γ_1 is a ‘don’t care’ function in the expression. At the end of the table another series of equivalent expressions is shown with γ_1 as ‘don’t care’. In that case the function 0 (all 0) and 15 (all 1) also have been applied.

Voluntary and mandatory “don’t care” functions

One of the questions to be asked is if the “don’t care” functions are voluntary. Can these functions be deleted without affecting the result? Or is their presence mandatory?

To solve this question the SISO expression:

$car3 \rightarrow \{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} \gamma_2 \{(c \rho_3 d) \gamma_3 (e \gamma_4 f)\}$ will be analyzed. Because an expression $(c \gamma_1 d)$ appears in the first part of the ‘car3’ expression as well as in the second it may be that this is a “don’t care” function in the first part. Assume that $(c \gamma_1 d)$ is “don’t care”. This means in the first part expression that: $\{(a \rho_1 b) \rho_2 (c \gamma_1 d)\}$ may be reduced to $(a \rho b)$. In this equation a, b, c and d are the constants and the functions are the variables.

If there are solutions where function ρ equals function ρ_1 then the voluntary “don’t care” functions have been found. In that case the expression ‘ $\rho_2 (c \gamma_1 d)$ ’ can be included or dropped from the expression without affecting the result. In all other cases, no matter how many equivalents are found they have to be included in the expression to arrive at the correct results. These “don’t care” functions are mandatory.

Some examples will be provided. One can start by searching all solutions where:

$$\{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} \gamma_2 \{(c \rho_3 d) \gamma_3 (e \gamma_4 f)\} = \{(a \rho_0 b)\} \gamma_2 \{(c \rho_3 d) \gamma_3 (e \gamma_4 f)\}$$

$\{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} \gamma_2 \{(c \rho_3 d) \gamma_3 (e \gamma_4 f)\} = \{(a \rho_0 b)\} \gamma_2 \{(c \rho_3 d) \gamma_3 (e \gamma_4 f)\}$				
ρ_0	γ_2	ρ_3	γ_3	γ_4
6	1	6	1	1
6	1	9	2	1
6	1	6	4	14
6	1	9	8	14
6	2	9	7	14
6	2	6	11	14
6	2	9	13	1
6	2	6	14	1
9	4	6	1	1
9	4	9	2	1
9	4	6	4	14
9	4	9	8	14
9	8	9	7	14
9	8	6	11	14
9	8	9	13	1
9	8	6	14	1

The previous table shows a set of solutions for the “don’t care” equation.

The next step is to find the expressions for which:

$$\{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} = (a \rho_0 b) \text{ with } \rho_1 = \rho_0 \text{ or the voluntary “don’t care” and}$$

$$\{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} = (a \rho_0 b) \text{ with } \rho_1 \neq \rho_0 \text{ or the mandatory “don’t care”}.$$

An example of the voluntary “don’t care”:

$\{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} = (a \rho_0 b)$			
ρ_0	ρ_1	ρ_2	γ_1
9	9	5	0
9	9	5	1
9	9	5	2
9	9	5	3
9	9	5	4
9	9	5	5
9	9	5	6
9	9	5	7
9	9	5	8
9	9	5	9
9	9	5	10
9	9	5	11
9	9	5	12
9	9	5	13
9	9	5	14
9	9	5	15

In this case one can either apply: $(a \ 9 \ b)$ or $\{(a \ 9 \ b) \ 5 \ (c \ \gamma_1 \ d)\}$ in the ‘car3’ expression without affecting the result. The function γ_1 is the “don’t care” function.

An example of the mandatory “don’t care”:

$\{(a \rho_1 b) \rho_2 (c \gamma_1 d)\} = (a \rho_0 b)$			
ρ_0	ρ_1	ρ_2	γ_1
9	6	10	0
9	6	10	1
9	6	10	2
9	6	10	3
9	6	10	4
9	6	10	5
9	6	10	6
9	6	10	7
9	6	10	8
9	6	10	9
9	6	10	10
9	6	10	11
9	6	10	12
9	6	10	13
9	6	10	14
9	6	10	15

In the case of the 'mandatory' "don't care" the whole expression has to be applied to arrive at the correct result. The "don't care" function here is also γ_1 .

Different cycles and different n-valued logics

The "don't care" effect (mandatory and voluntary) will be enhanced when one expands the cycle for creating 'equivalent' expressions to 8 inputs. These expressions will have 8 terms ($x \rho w$), which are connected by 7 functions. These 8-inputs-single-output expressions apply 15 functions. They have significant redundancy which will be expressed in a greater number of "don't care" functions compared to the SISO equivalence.

Elsewhere an analysis has been done by the author on 3-valued full adders. The number of possible 3-valued functions is over 19,000. The number of equivalent expressions for 3-valued logic based full adders is exponentially greater than for binary full adders.

Ternary and n-valued equivalent expressions as applied in this article will show many, many more "don't care" functions and "don't care" expressions, which in a way may be characterized as 'junk' logic or as 'junk' logic with a purpose.

About Ternarylogic LLC: Ternarylogic LLC is a New Jersey based R&D company. Its mission is to create novel MVL technology. The company owns a portfolio of inventions related to scramblers/descramblers, sequence generators and sequence detectors, sequence correlators, gates and inverters based circuitry, non-binary multipliers, latches and other non-volatile memory elements, optical disk applications and MC-DSSS technology.

© 2005, Ternarylogic LLC. All rights reserved.