

# An Intriguing Digital Logic Problem: How fundamental is an inverter?

Srinivasan Ramasubramanian  
 Department of Electrical and Computer Engineering  
 University of Arizona, Tucson, AZ 85721  
 srini@ece.arizona.edu

Thursday May 3, 2007

## I. BACKGROUND

The first question in my PhD qualifying exam (held in the ECE Department at Iowa State University in Spring 1999) was:

*Given three binary inputs  $A$ ,  $B$ , and  $C$ , obtain their complements  $\overline{A}$ ,  $\overline{B}$ , and  $\overline{C}$  using only two inverters and any number of AND and OR gates.*

This deceptively simple question came from Late Prof. Charlie Wright (ECE, Iowa State University). Clearly, none of us solved the problem in the examination. Yet, we were curious to know what the solution was. We discussed this problem with our advisor Prof. Arun K. Somani (ECE, Iowa State University) and we got the solution method. The key is to compute the number of 1s in the input and encode that as a binary number! Then, using this encoded input, we can obtain the individual complements. Once this method was revealed, we wrote down the solution once completely to verify it. Over the years, there were several instances when I remembered this problem and I admired the sheer elegance of the solution.

As recently as last week, I remembered the problem when we were asked to provide questions for the PhD qualifying exam in the ECE department at the University of Arizona. Upon searching the Internet to see how popular this question has become over the years, I found an EE Times article [EE] and a follow-up discussion on Programmable Logic Design Line [PLD-1] that discusses solutions from several readers and identifies the key to the solution in the end. The second follow-up discussion [PLD-2] discusses hardware-based solutions.

I found it interesting that this problem was relatively unknown to many digital designers even as of May 2006. I shared the articles with my PhD advisor and two other colleagues of mine (back when I was a graduate student), Dr. Sashisekaran Thiagarajan (Ciena Corporation) and Dr. Murari Sridharan (Microsoft Corporation, who also took the same qualifying examination). Sashi mentioned that a generalization of the above question would be even more interesting. Although I have thought about it myself over the years and was sort of convinced myself of what the answer should be, I never formally wrote the solution method. Yesterday, I decided to spend some time to write down the generalized solution method in a formal way.

## II. PROBLEM STATEMENT

Given  $n$  binary inputs,  $X = (x_{n-1}, x_{n-2}, \dots, x_1, x_0)$ , compute the complement  $\overline{X} = (\overline{x_{n-1}}, \overline{x_{n-2}}, \dots, \overline{x_1}, \overline{x_0})$  with minimum number of inverters. The only other gates allowed are AND and OR and any number of them may be employed.

## III. A SOLUTION

Let  $K$  denote the number of bits required to represent the integer  $n$  in binary representation,  $K = \lfloor \log_2(n) \rfloor + 1$ . The method to invert  $n$  inputs described here requires exactly  $K$  inverters.

**Step 1:** Let the  $B = (b_{K-1}, b_{K-2}, \dots, b_1, b_0)$  denote the binary representation of the number of 1s in the input – represented as a  $K$ -bit vector. Let  $\overline{B} = (\overline{b_{K-1}}, \overline{b_{K-2}}, \dots, \overline{b_1}, \overline{b_0})$ . The bits in  $B$  are computed in an ordered fashion starting from the most significant bit. A bit  $b_i$  is computed using an *unate* function involving the inputs along with the bits  $b_j$  and  $\overline{b_j}$  for all  $j > i$ . Upon computing  $b_i$ , the complement  $\overline{b_i}$  is obtained using an inverter. The details of computing the individual bits is as follows.

Let  $M_w$  ( $w = \{1, 2, \dots, n\}$ ) denote whether there are at least  $w$  1s or not in the input, 1 if true and 0 otherwise.  $M_w$  is the disjunction of all possible  $w$ -conjunctions<sup>1</sup> from the set of  $n$  inputs, hence is an unate function. The bit  $b_{K-1}$  denotes if the set of inputs has at least  $2^{K-1}$  1s or not. Therefore, bit  $b_{K-1}$  is computed as:

$$b_{K-1} = M_{2^{K-1}} \quad (1)$$

By employing an inverter, the complement is obtained.

$$\overline{b_{K-1}} = \text{INVERT}(b_{K-1})$$

<sup>1</sup>A  $w$ -conjunction is the conjunction of  $w$  input variables from a given set of binary variables.

Bit  $b_{K-2}$  is computed as:

$$b_{K-2} = b_{K-1}M_w + \overline{b_{K-1}}M_{w'} \quad (2)$$

where  $w = 2^{K-1} + 2^{K-2}$  and  $w' = 2^{K-2}$ . The bit  $b_{K-2}$  is 1 if the inputs contain at least  $w$  1s. If not, it should contain at least  $w'$  1s. Otherwise,  $b_{K-2}$  is 0. Again, at the expense of an inverter, the complement may be obtained.

$$\overline{b_{K-2}} = \text{INVERT}(b_{K-2})$$

Now,  $b_{K-3}$  may be computed as:

$$b_{K-3} = b_{K-1}b_{K-2}M_w + b_{K-1}\overline{b_{K-2}}M_{w'} + \overline{b_{K-1}}b_{K-2}M_{w''} + \overline{b_{K-1}}\overline{b_{K-2}}M_{w'''}$$

where

$$\begin{aligned} w &= 2^{K-1} + 2^{K-2} + 2^{K-3} \\ w' &= 2^{K-1} + 2^{K-3} \\ w'' &= 2^{K-2} + 2^{K-3} \\ w''' &= 2^{K-3}. \end{aligned}$$

The remaining bits and their complements may be iteratively computed in a similar fashion. Note that based on the above expression, the computation of bit  $b_i$  would consider all combinations involving the higher bits,  $b_{i+1}$  through  $b_{K-1}$  and their complements. Thus, the computation of  $B$  and  $\overline{B}$  requires exactly  $K$  inverters.

**Step 2:** Given that  $B$  and  $\overline{B}$  have been computed, the complement of the individual inputs may be obtained without requiring any additional inverters. Let  $V_w$  ( $w = \{0, 1, \dots, n\}$ ) be a binary variable that is set to 1 if the number of 1s in the input is  $w$ , and 0 otherwise.  $V_w$  may be computed from the bits of  $B$  and  $\overline{B}$  without requiring any additional inverters.

Let  $X_i$  ( $i = \{1, 2, \dots, n\}$ ) denote the set of inputs excluding input  $x_i$ , i.e.  $X_i = (x_{n-1}, x_{n-2}, \dots, x_{i+1}, x_{i-1}, \dots, x_1, x_0)$ . Let  $C_{iw}$  ( $w = \{1, \dots, n-1\}$ ) denote if the set  $X_i$  has at least  $w$  1s or not.  $C_{iw}$  the disjunction of all possible  $w$ -conjunctions of the elements from the set  $X_i$ . Note that  $C_{iw}$  is also an unate function for all  $i$  and  $w$ . The complement of input  $x_i$  is then computed as:

$$\overline{x_i} = V_0 + \sum_{w=1}^{n-1} V_w C_{iw} \quad (3)$$

where the summation represents the binary OR operation and product represents the binary AND operation. For a given set of inputs with  $w$  1s, only  $V_w$  will be 1. Therefore, the term  $C_{iw}$  simply identifies if there are  $w$  1s without considering input  $x_i$ . If yes,  $x_i$  must be 0, hence  $\overline{x_i} = 1$ . If not, then  $x_i$  must be 1, hence  $\overline{x_i} = 0$ .

**Relation to solution by Hadar:** Note that if the number of inputs is 3, then it is sufficient to have two bits to represent the number of 1s. The bits  $b_1$  and  $b_0$  are the same as the  $R$  and  $S$  bits, respectively, in the solution by Hadar [PLD-1].

#### IV. CONCLUSION

This article shows an approach to compute the complement of  $n$  binary inputs with  $K$  inverters, where  $K = \lfloor \log_2(n) \rfloor + 1$ . The drawback is that the number of AND/OR gates required is exponential in the number of inputs. While the problem may not have any significant practical impact, it is a beautiful fundamental question that picks the curiosity of anyone interested in digital logic design.

I do not think this problem can be answered by someone in a PhD qualifying exam (say, given an entire hour for this problem alone) if that's the first time the person is seeing the question. However, as my PhD advisor commented, the reason this question was given in our qualifying exam was probably very different – if you cannot find a solution to a problem in a reasonable time, move on to other questions. It took 30 minutes for me to move on to the next question. I managed to pass the qualifying examination even though this question was lingering in the back of my mind during the entire examination.

#### ACKNOWLEDGMENTS

Many thanks to Late Prof. Charlie Wright (ECE, Iowa State University) for including the problem in our PhD qualifying examination. Thanks to Prof. Arun K. Somani (ECE, Iowa State University) for discussions (in 1999) on the solution method for inverting three binary inputs with two inverters. Thanks to Sashi for bringing up the generalization part of the problem.

#### REFERENCES

- [EE] **Logically Speaking Column from EE Times:** <http://www.eetimes.com/showArticle.jhtml;?articleID=187002658>
- [PLD-1] **Programmable Logic Design Line Column 1:** <http://www.pldesignline.com/187202855>
- [PLD-2] **Programmable Logic Design Line Column 2:** <http://www.pldesignline.com/188101054>

```

// Filename: inverter5bit.v
// Author : Srinivasan Ramasubramanian
// Function: Inverts 5 binary inputs: x0 through x4.
// Output  : x0comp through x4comp -- the inverted inputs.
//          : b2, b1, b0 -- three bits that encodes the # of 1s in the input.

module inverter5bit(x0, x1, x2, x3, x4, x0comp, x1comp, x2comp, x3comp, x4comp, b0, b1, b2);
input x0, x1, x2, x3, x4;
output x0comp, x1comp, x2comp, x3comp, x4comp;
output b0, b1, b2;

wire m0, m1, m2, m3, m4, m5;
wire c01, c02, c03, c04;
wire c11, c12, c13, c14;
wire c21, c22, c23, c24;
wire c31, c32, c33, c34;
wire c41, c42, c43, c44;
wire v0, v1, v2, v3, v4, v5;
wire b0, b1, b2, b0comp, b1comp, b2comp;

assign m1 = x0 | x1 | x2 | x3 | x4;

assign m2 = (x0 & x1) | (x0 & x2) | (x0 & x3) | (x0 & x4) |
            (x1 & x2) | (x1 & x3) | (x1 & x4) |
            (x2 & x3) | (x2 & x4) |
            (x3 & x4) ;

// Use the same terms from m2; but use the missing terms
// that way it's easy to see that the combinations are correct.
assign m3 = (x2 & x3 & x4) | (x1 & x3 & x4) | (x1 & x2 & x4) | (x1 & x2 & x3) |
            (x0 & x3 & x4) | (x0 & x2 & x4) | (x0 & x2 & x3) |
            (x0 & x1 & x4) | (x0 & x1 & x3) |
            (x0 & x1 & x2) ;

assign m4 = (x1 & x2 & x3 & x4) | (x0 & x2 & x3 & x4) |
            (x0 & x1 & x3 & x4) | (x0 & x1 & x2 & x4) | (x0 & x1 & x2 & x3);

assign m5 = (x0 & x1 & x2 & x3 & x4);

// Computing bi -> The bits in the encoded binary value for the # of 1s
assign b2 = m4;

assign b2comp = ~b2;

assign b1 = (b2comp & m2);
// (b2 & m6) --m6 is obviously 0 as we have only 5 inputs

assign b1comp = ~b1;

assign b0 = (b2 & b1comp & m5) | (b2comp & b1 & m3) | (b2comp & b1comp & m1);
// (b1 & b2 & m7) = 0;

assign b0comp = ~b0;

// Computing Vi - indicating if there are i ones or not.

assign v0 = b2comp & b1comp & b0comp;
assign v1 = b2comp & b1comp & b0;
assign v2 = b2comp & b1 & b0comp;
assign v3 = b2comp & b1 & b0;
assign v4 = b2 & b1comp & b0comp;
assign v5 = b2 & b1comp & b0; // for sake of completeness

```

```

// not used to compute any of the complements.

// Computing C0s and x0comp;
// without using x0, find all combinations.
assign c01 = x1 | x2 | x3 | x4;
assign c02 = (x1 & x2) | (x1 & x3) | (x1 & x4) |
             (x2 & x3) | (x2 & x4) |
             (x3 & x4) ;
assign c03 = (x2 & x3 & x4) | (x1 & x3 & x4) | (x1 & x2 & x4) | (x1 & x2 & x3);
assign c04 = (x1 & x2 & x3 & x4);

assign x0comp = v0 | (v1 & c01) | (v2 & c02) | (v3 & c03) | (v4 & c04);

// Computing C1s and x1comp
assign c11 = x2 | x3 | x4 | x0;
assign c12 = (x2 & x3) | (x2 & x4) | (x2 & x0) |
             (x3 & x4) | (x3 & x0) |
             (x4 & x0) ;
assign c13 = (x3 & x4 & x0) | (x2 & x4 & x0) | (x2 & x3 & x0) | (x2 & x3 & x4);
assign c14 = (x2 & x3 & x4 & x0);

assign x1comp = v0 | (v1 & c11) | (v2 & c12) | (v3 & c13) | (v4 & c14);

// Computing C2s and x2comp
assign c21 = x3 | x4 | x0 | x1;
assign c22 = (x3 & x4) | (x3 & x0) | (x3 & x1) |
             (x4 & x0) | (x4 & x1) |
             (x0 & x1) ;
assign c23 = (x4 & x0 & x1) | (x3 & x0 & x1) | (x3 & x4 & x1) | (x3 & x4 & x0);
assign c24 = (x3 & x4 & x0 & x1);

assign x2comp = v0 | (v1 & c21) | (v2 & c22) | (v3 & c23) | (v4 & c24);

// Computing C3s and x3comp
assign c31 = x4 | x0 | x1 | x2;
assign c32 = (x4 & x0) | (x4 & x1) | (x4 & x2) |
             (x0 & x1) | (x0 & x2) |
             (x1 & x2) ;
assign c33 = (x0 & x1 & x2) | (x4 & x1 & x2) | (x4 & x0 & x2) | (x4 & x0 & x1);
assign c34 = (x4 & x0 & x1 & x2);

assign x3comp = v0 | (v1 & c31) | (v2 & c32) | (v3 & c33) | (v4 & c34);

// Computing C4s and x4 comp
assign c41 = x0 | x1 | x2 | x3;
assign c42 = (x0 & x1) | (x0 & x2) | (x0 & x3) |
             (x1 & x2) | (x1 & x3) |
             (x2 & x3) ;
assign c43 = (x1 & x2 & x3) | (x0 & x2 & x3) | (x0 & x1 & x3) | (x0 & x1 & x2);
assign c44 = (x0 & x1 & x2 & x3);

assign x4comp = v0 | (v1 & c41) | (v2 & c42) | (v3 & c43) | (v4 & c44);

endmodule
// End of 5-bit inverter module.

```

```

// Filename: inverter4bit.v
// Author : Srinivasan Ramasubramanian
// Function: Inverts 4 binary inputs: x0 through x3.
// Output : x0comp through x3comp -- the inverted inputs.
// : b2, b1, b0 -- three bits that encodes the # of 1s in the input.

module inverter4bit(x0, x1, x2, x3, x0comp, x1comp, x2comp, x3comp, b0, b1, b2);
input x0, x1, x2, x3;
output x0comp, x1comp, x2comp, x3comp;
output b0, b1, b2;

wire m0, m1, m2, m3, m4;
wire c01, c02, c03;
wire c11, c12, c13;
wire c21, c22, c23;
wire c31, c32, c33;
wire c41, c42, c43;
wire v0, v1, v2, v3, v4;
wire b0, b1, b2, b0comp, b1comp, b2comp;

assign m1 = x0 | x1 | x2 | x3;

assign m2 = (x0 & x1) | (x0 & x2) | (x0 & x3) |
            (x1 & x2) | (x1 & x3) |
            (x2 & x3) ;

// Use the same terms from m2; but use the missing terms
// that way it's easy to see that the combinations are correct.
assign m3 = (x1 & x2 & x3) | (x0 & x2 & x3) | (x0 & x1 & x3) |
            (x0 & x1 & x2) ;

assign m4 = (x0 & x1 & x2 & x3);

// Computing bi -> The bits in the encoded binary value for the # of 1s
assign b2 = m4;

assign b2comp = ~b2;

assign b1 = (b2comp & m2);
// (b2 & m6) --m6 is obviously 0 as we have only 5 inputs
assign b1comp = ~b1;

assign b0 = (b2comp & b1 & m3) | (b2comp & b1comp & m1);
// m5 and m7 are zeros.

assign b0comp = ~b0;

// Computing Vi - indicating if there are i ones or not.

assign v0 = b2comp & b1comp & b0comp;
assign v1 = b2comp & b1comp & b0;
assign v2 = b2comp & b1 & b0comp;
assign v3 = b2comp & b1 & b0;
assign v4 = b2 & b1comp & b0comp; // not used in computation of complements.

// Computing C0s and x0comp;
// without using x0, find all combinations.
assign c01 = x1 | x2 | x3;
assign c02 = (x1 & x2) | (x1 & x3) | (x2 & x3);
assign c03 = (x1 & x2 & x3);

```

```
assign x0comp = v0 | (v1 & c01) | (v2 & c02) | (v3 & c03);

// Computing C1s and x1comp
assign c11 = x2 | x3 | x0;
assign c12 = (x2 & x3) | (x2 & x0) | (x3 & x0) ;
assign c13 = (x2 & x3 & x0);

assign x1comp = v0 | (v1 & c11) | (v2 & c12) | (v3 & c13);

// Computing C2s and x2comp
assign c21 = x3 | x0 | x1;
assign c22 = (x3 & x0) | (x3 & x1) | (x0 & x1) ;
assign c23 = (x3 & x0 & x1);

assign x2comp = v0 | (v1 & c21) | (v2 & c22) | (v3 & c23);

// Computing C3s and x3comp
assign c31 = x0 | x1 | x2;
assign c32 = (x0 & x1) | (x0 & x2) | (x1 & x2) ;
assign c33 = (x0 & x1 & x2);

assign x3comp = v0 | (v1 & c31) | (v2 & c32) | (v3 & c33);

endmodule
// End of 4-bit inverter module
```