Implementation of a Two's Complement Circuit*

*Note: Project of the course ICE3405P-Computer Organization

Nan Lin
SPEIT
Shanghai Jiao Tong University
Shanghai, China
lns_brandon@sjtu.edu.cn

Yanxu Meng
SPEIT
Shanghai Jiao Tong University
Shanghai, China
email address or ORCID

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Question 1-5

A. Question 1

The two's complement of a binary number is obtained by following the rule:

- If the sign bit (highest bit) is 0, it indicates that the sign-magnitude representation is a positive number. In this case, the binary number is unchanged.
- If the sign bit is 1, we should invert its digits and add one to the least significant bit.

This method allows for binary arithmetic and simplifies the design of digital circuits for arithmetic operations. A straightforward implementation is to adopt different operation modes depending on the sign bit using a multiplexer. A typical multiplexer is shown in Figure 1. For an 8-bit binary number, suppose that it could be represented as:

$$\overline{s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0} \tag{1}$$

s_7 is the sign bit, therefore playing the role of Select for all other bits (s_0 to s_6) here.

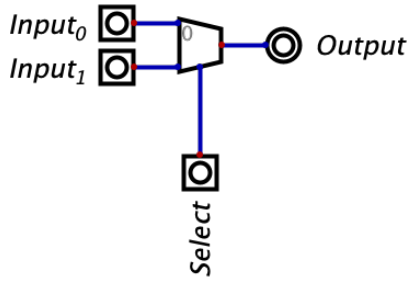

Fig. 1. A Multiplexer. When Select is 0, the output corresponds to the value of Input_0, neglecting the condition of Input_1. Inversely, when Select is 1, the output corresponds to the value of Input_1.

In order to add one to the least significant bit, an adder is required. A half-adder is a fundamental component in digital arithmetic. It takes two single-bit binary inputs and produces a sum and carry output. The truth table for a half-adder is shown below:

A	B	Sum (Result)	CarryOut
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

TABLE I
TRUTH TABLE FOR A HALF-ADDER

The corresponding half-adder circuit diagram is illustrated in Figure 2.

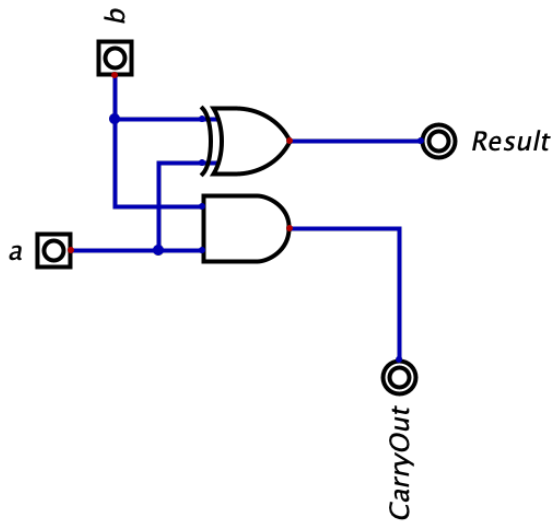

Fig. 2. Half-Adder Circuit

We use a half-adder rather than a full adder for multiple reasons:

- A half-adder has a relatively simpler structure. Most importantly, its depth is one less than a full adder and it contains fewer gates which provides the advantage of lower energy consumption and less time consumption.
- A half-adder is proved to be effective because to implement a two's complement circuit, we only need to calculate $\overline{s_6 \dots s_0} + \overline{0 \dots 01}$ (taking 8-bit binary numbers

as an example). Since all the bits are 0, we can put the CarryOut onto the place of another adder.

To implement an 8-bit two's complement circuit, we combine multiple half-adders; each half-adder is used to calculate the corresponding bit of the binary number. The process involves inverting each bit of the input number and then adding one using a series of adders. The detailed circuit diagram is shown in Figure 3.

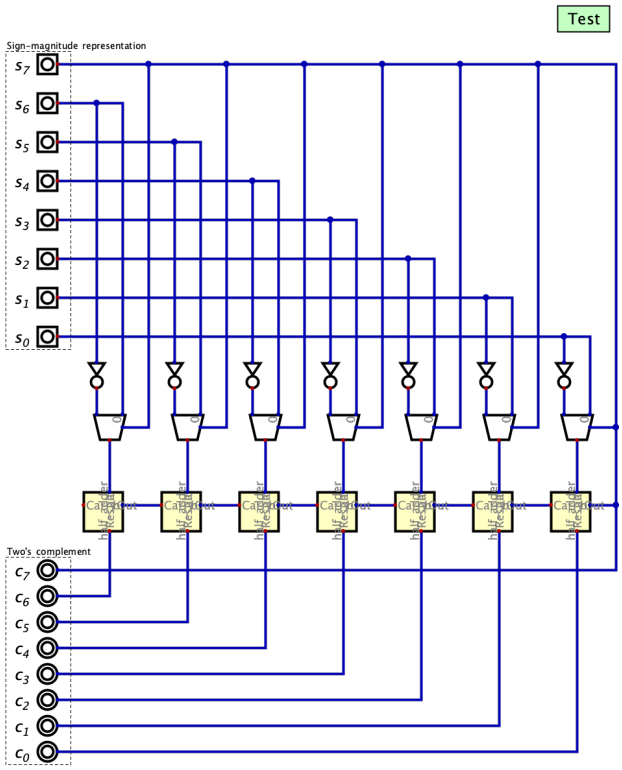

Fig. 3. 8-bit Two's Complement Circuit. The yellow box is a half-adder implemented in Figure 2

B. Question 2

To analyze the depth and complexity of the two's complement circuit for 8 bits, we divide the process into two stages.

The first stage is the **inversion stage**. Each bit requires a NOT gate, thus in the first stage:

- Depth: 1 (because all of the NOT gates operate simultaneously)
- Complexity: 7 (NOT gates)

The second stage is the **addition stage**. For each bit, a half-adder is applied. The depth of each half-adder is 1 and the complexity is 2. Since there are 7 bits that need to be calculated one by one, in the second stage:

- Depth: $7 \times 1 = 7$
- Complexity: $7 \times 2 = 14$

Summing up the two stages, the depth of calculation of the two's complement of an 8-bit binary number is $1+7 = 8$ and the complexity is $7 + 14 = 21$.

We expand the results to any 2^p-bit ($p \in \mathbb{N}$) two's complement circuit.

1) In the **inversion stage**, the depth is 1 due to parallelization, and the complexity is $2^p - 1$.
2) In the **addition stage**, the depth is $2^p - 1$ and the number of gates required is $2 \times (2^p - 1)$.

Conclusion: For a 2^p-bit two's complement circuit, the depth is 2^p, and the complexity is $3(2^p - 1)$.

C. Question 3

In order to reduce the depth of the circuit, having a closer look at the circuit implemented in Section I-A, the main problem is that the CarryOut is passed on in series. In other words, to calculate the two's representation of s_k of a 2^p-bit binary number ($0 < k \leq 2^p - 2$), we have to wait for the values of $k - 1$ CarryOuts one by one.

To optimize this process, we propose a new version of the two's complement circuit for an 8-bit machine by employing a method that uses two 4-bit lookahead carry adders in series. This approach allows for the calculation of four bits and their carry at once, then passing the carry to the next set of four bits. This reflects the divide and conquer strategy.

The implementation involves the following steps:

1) **Inversion Stage**: This stage is unchanged, where each of the 8 bits is inverted using NOT gates.
2) **Addition Stage**: The 8-bit addition is divided into two 4-bit additions using lookahead carry adders.
   - **First 4-bit Adder**: Computes the result and the carry for the first four bits.
   - **Second 4-bit Adder**: Computes the result for the next three (but not four!) bits.

We then implement a classical 4-bit lookahead carry adders which could be found in textbooks. First we introduce two fundamental components.

In digital circuits, the Generate-Propagate (GP) logic is used to speed up the carry calculation in adders. For two binary inputs A_i and B_i, the generate (G_i) and propagate (P_i) signals are defined as:

$$g_i = a_i \cdot b_i \tag{2}$$

$$p_i = a_i + b_i \tag{3}$$

The GP generator is used to generate respectively the AND and OR result of two variables, this result will be deploited later on. The implementation of the GP generator is displayed in Figure 4. For any bit, the value is

$$\forall k \in [\![1, 2^p - 2]\!], \quad s_k = a_k + b_k + c_{k-1} \tag{4}$$

where c_{k-1} is the carry bit into position k.

The carry bits could be calculated as follows:

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i \tag{5}$$

owing to the reason that the carry bit is 1 if

- Both of the two bits are 1

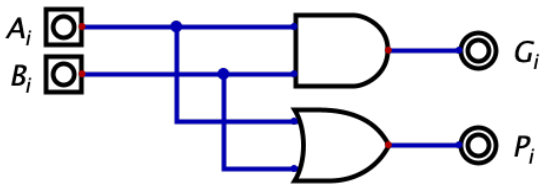

Fig. 4. GP Generator Circuit

- Any of the two bits is 1 and the carry bit from the former bit is 1.

Here we replace $a_i b_i$ and $a_i + b_i$ with g_i and p_i. The Equation 5 turns into:

$$c_{i+1} = g_i + p_i c_i \tag{6}$$

We therefore deduce all the carry bits based on Equation 6:

$$c_1 = g_0 + p_0 c_0 \tag{7}$$
$$c_2 = g_1 + p_1 c_1 = g_1 + p_1 g_0 + p_1 p_0 c_0 \tag{8}$$
$$c_3 = g_2 + p_2 c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \tag{9}$$
$$c_4 = g_3 + p_3 c_3 = g_3 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \tag{10}$$

A full adder also takes the CarryIn of the formal bit compared to a half adder. It is displayed in Figure 5.

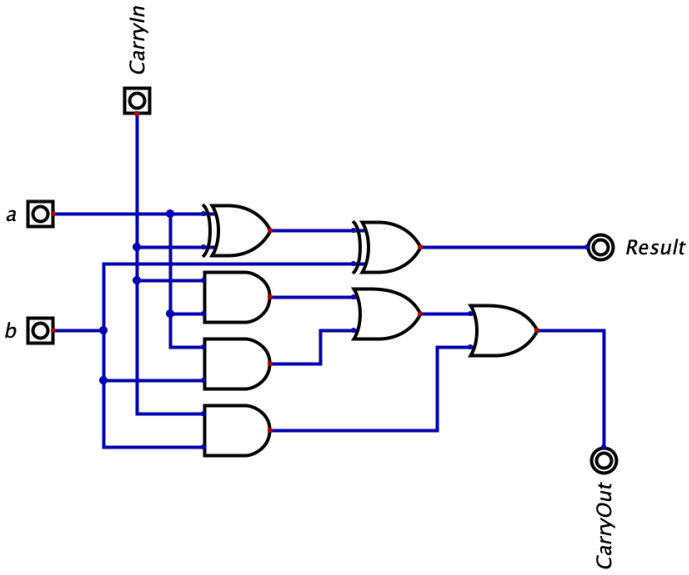

Fig. 5. Full-Adder Circuit

A classic 4-bit lookahead carry adder based on full-adders is shown in Figure 6. We then combine the two 4-bit adder in series. The final two's complement circuit is shown in Figure 7. When the sign bit is 1, it implies that we should invert its digits and add one to the least significant bit. In this case, the sign bit is passed to the first CarryIn input, and the operation of the full adder is identical to:

$$\overline{a_3 a_2 a_1 a_0} + \overline{0000} \quad \text{with CarryIn 1} \tag{11}$$
$$\overline{0 a_6 a_5 a_4} + \overline{0000} \quad \text{with CarryIn from the former 4-bit adder} \tag{12}$$

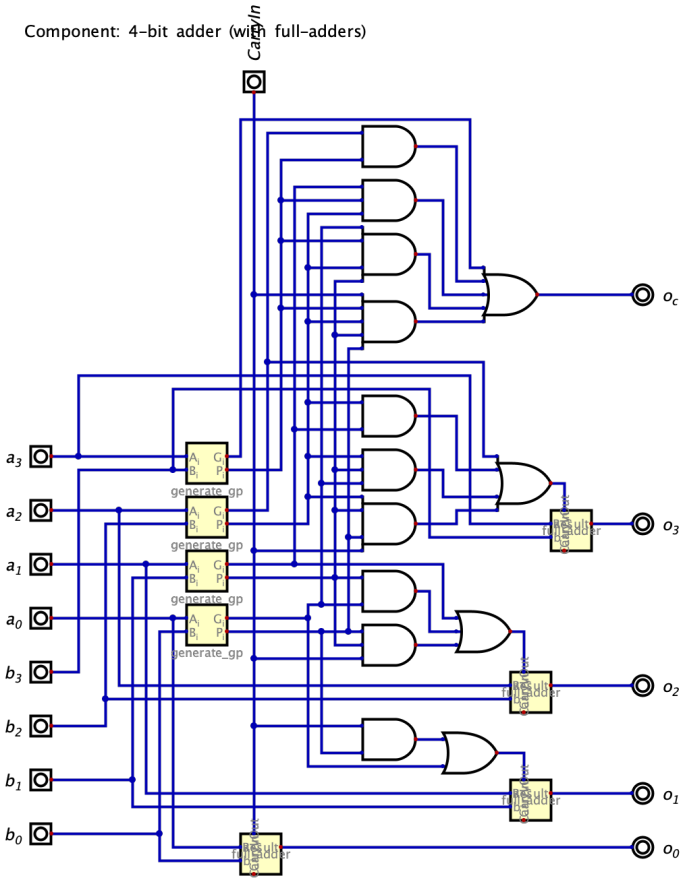

Fig. 6. 4-bit Lookahead Carry Adder based on Full-Adders

The reason why we put in series of 4-bit Lookahead Carry Adder instead of series of 8 (or even 16, 32, ...)-bit Lookahead Carry Adder is to reduce the complexity of designing the circuit. As shown in Figure fig:lca, to implement a 4-bit Lookahead Carry Adder, we need an AND gate with 5 inputs. This is already hard to fabricate in reality and it consume much more time and energy if we separate the AND gates.

However, there are still spaces of optimization. This indeed is the implementation of a standard additionner between two 8-bit binary numbers. Here, we're only required to perform addition on binary numbers with 1. We could throw out 0s and full-adders and adopt half-adders.

Furthermore, since the other adder is always 0 in Equation 11, the G(enerate) signal is always 0. This implies for any valid value of g_i, they should be 0. Equations 7 to 10

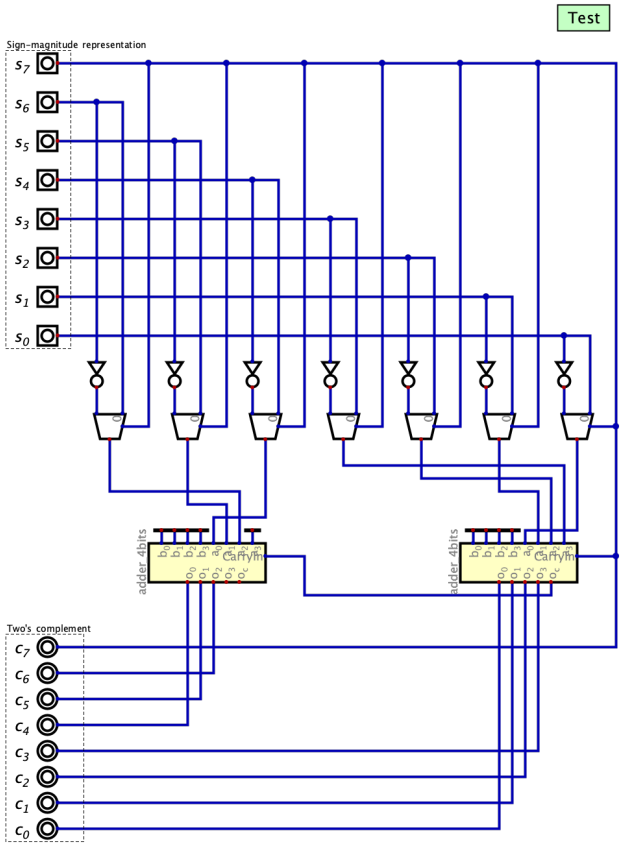

Fig. 7. 8-bit Two's Complement Circuit (Divide and Conquer Method) Based on Full-Adders

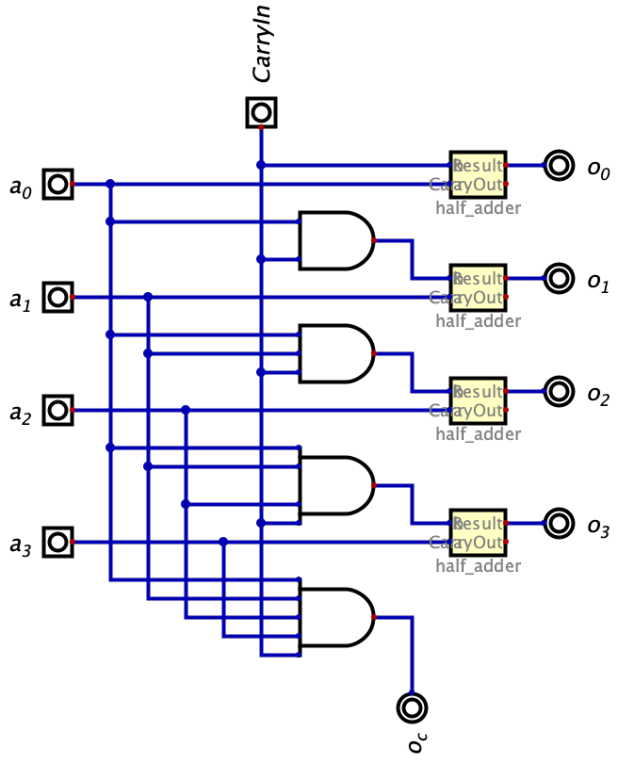

Fig. 8. Optimized 4-bit Lookahead Carry Adder based on Half-Adders and simplified gates

could be transformed into:

$$c_1 = p_0 c_0 \tag{13}$$

$$c_2 = p_1 p_0 c_0 \tag{14}$$

$$c_3 = p_2 p_1 p_0 c_0 \tag{15}$$

$$c_4 = p_3 p_2 p_1 p_0 c_0 \tag{16}$$

Based on these two points, the 4-bit Lookahead Carry Adder could be optimized as shown in Figure 8.

The corresponding 8-bit two's complement circuit is shown in Figure 9.

D. Question 4

For a 8-bit two's complement circuit, to calculate the depth and complexity of the circuit, we divide the circuit into two parts:

1) In the **inversion stage**, all components are inversed simultaneously, thus depth is 1 and amount of NOT gates required is 7.
2) In the **addition stage**, the two 4-bit lookahead carry adder one after another. They are connected in series. For a single 4-bit lookahead, its depth is 2 since one for the AND gates and one for the Half-Adders. The number of gates utilized is:

$$4 + 4 \times 2 = 12 \tag{17}$$

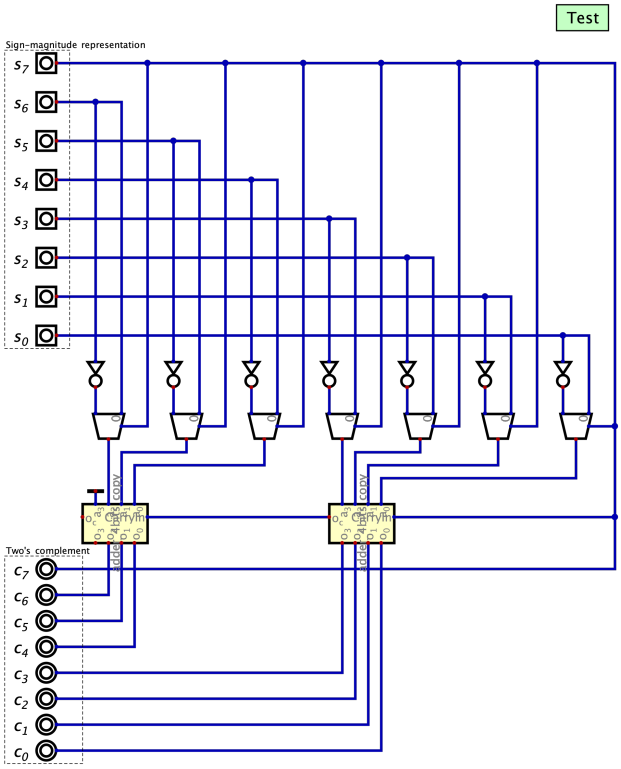

Fig. 9. Optimized 8-bit Two's Complement Circuit (Divide and Conquer Method) Based on 4-bit Lookahead Carry Adder Proposed in Figure 8

Since there are two lookahead carry adders, therefore, its depth is 4 and its complexity is 24.

Therefore, for a 8-bit circuit, its depth is $1+4=5$ and its complexity is $7+24=31$.

Now, for any 2^p-bit two's complement circuit, the inversion stage is unchanged: the depth is 1 and the complexity is 2^p-1 in this stage. We need to count how many 4-bit Lookahead adders needed.

The main concept is to divide the large binary number into groups of 4-bit binary numbers. After that, we put them into series. The amount of groups:

$$\begin{cases} \frac{2^p}{4} = 2^{p-2} & \text{if} \quad p \geq 2 \\ 1 & \text{if} \quad p = 0,\ 1 \end{cases} \tag{18}$$

Therefore, The depth is $2\times 2^{p-2} = 2^{p-1}$ and the complexity is $12 \times 2^{p-2} = 3 \times 2^p$ in the addition stage.

Conclusion: The depth is $2^{p-1}+1$ and the complexity is $2^p - 1 + 3 \times 2^p = 2^{p+2} - 1$.

(?????????????????????? REALLY ??????????????????????)

E. Question 5

We export the VHDL code based on the circuit designed in the Digital software and generate the code using gtkwave. Reminding that the input 8-bit binary number and the output 8-bit binary number are represented by:

$$\overline{s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0} \longrightarrow \overline{c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0} \tag{19}$$

The text data is listed in Figure 10.

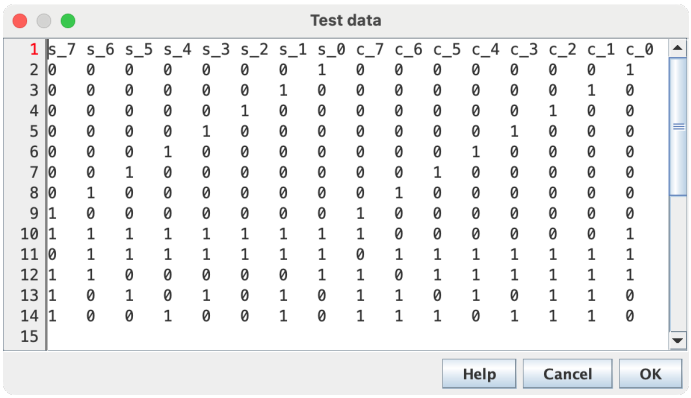

Fig. 10. Test Benchmark

As is required, all the ports/input-output have a delay of 1 unit time and all the multiplexer have a delay of 2 unit time. The corresponding simulation of signals is shown in Figure 14.

Analysis of the result:

- Setting the delay of the multiplexer as 2 and the delay of the NOT gate as 1 allow us to directly exploit the result of the NOT gate after the result is calculated. (Figure 12)
- The calculation of the two 4-bit Lookahead Carry Adder proved the fact the two adders are connected in series. One would begin to operate only after the other hass finished its calculation. (Figure 13)

(An equivilent expression is shown in Figure 11. It is worthy to note that this code cannot generate VHDL code for the reason that the delay component is not implemented in VHDL (at least not in Digital software))

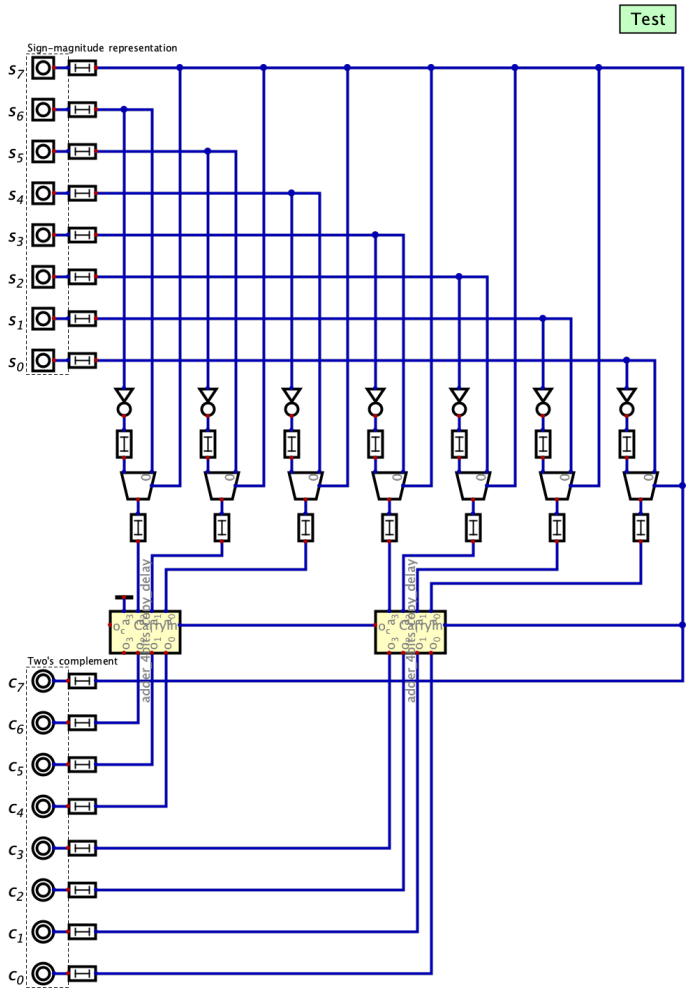

Fig. 11. Equivalent Representation of the circuit combined with delated I/O, Gates and Multiplexers. This Representation is also useful to understand the Analysis 1. Part

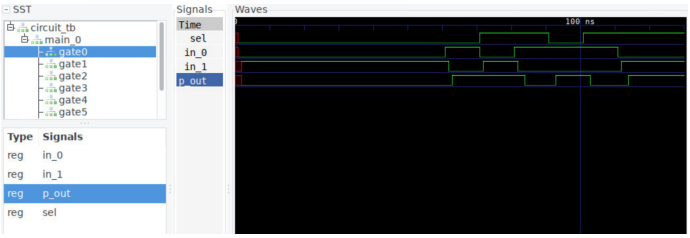

Fig. 12. Analysis 1: Observation of the time delay of the component NOT Gate + Multiplexer. After the signal is passed to the NOT Gate, the NOT Gate generate the result after 1 ns, and the multiplexer generate the result after 2 ns, which is time-efficient

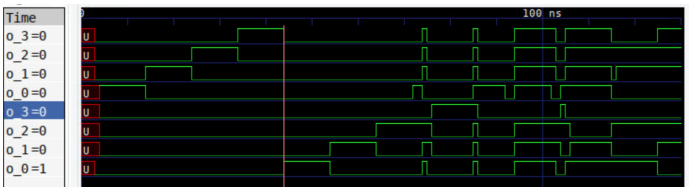

Fig. 13. Analysis 2: Observation of the time delay of the two 4-bit Lookahead Carry Adder. The upper four signals correspond to the lower 4-bits of the output. (Note that here, o_3 has no actual meaning since the highest bit should be the sign bit)

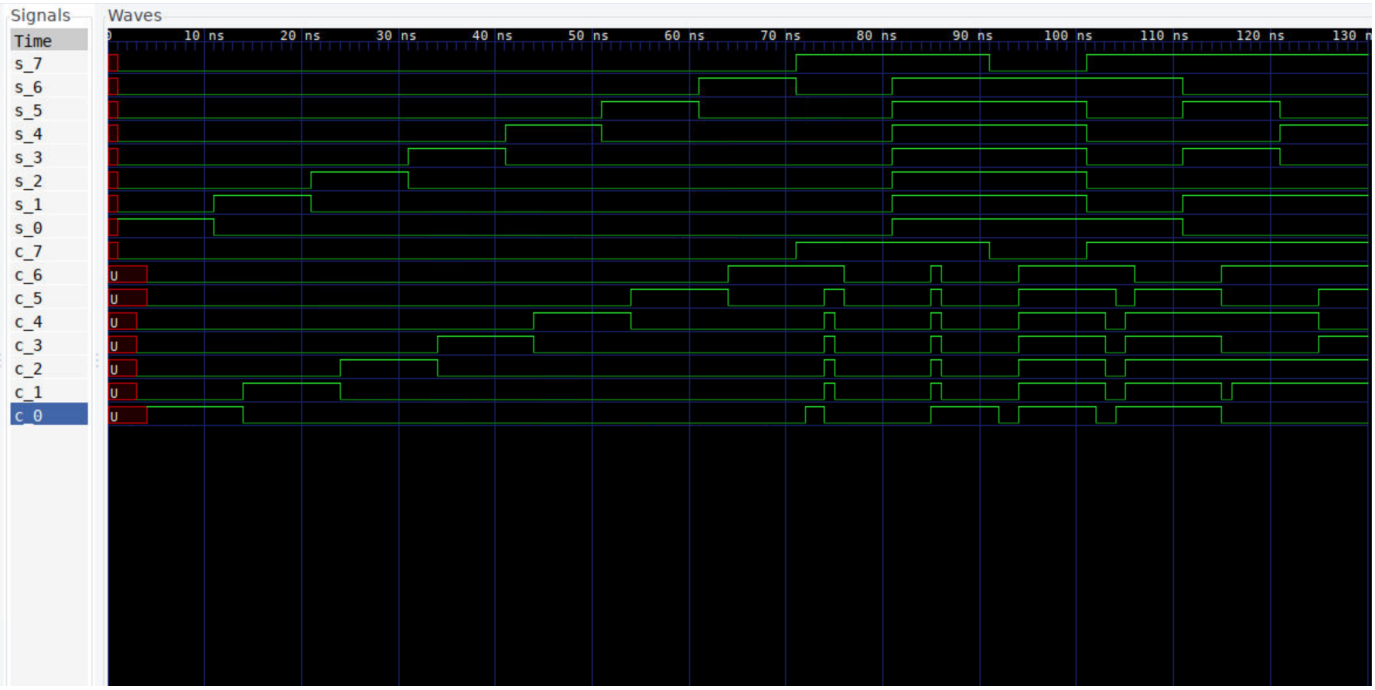

Fig. 14. Result with delay with test data in Figure 10