ANNALS OF THE FACULTY OF ENGINEERING HUNEDOARA
2006, Tome IV, Fascicole 1, (ISSN 1584 – 2665)
FACULTY OF ENGINEERING HUNEDOARA,
5, REVOLUTIEI, 331128, HUNEDOARA

USE THE GRAPHICAL PROGRAMMING LANGUAGE LABVIEW FOR LEARN THE BASICS OF PARALLEL ADDERS

CRISTEA Ana Daniela

UNIVERSITY POLITEHNICA TIMISOARA
FACULTY OF ENGINEERING HUNEDOARA

ABSTRACT:
This article propose to realize:
- Explein how to use LabVIEW to bild a simulation;
- how to implement two tips of adders: carry selection adder and carry completion adder;

KEYWORDS:
Adder, carry , LabVIEW, programming

1. INTRODUCTION

LabVIEW is a entirely graphical language which looks somewhat like an electronic schematic diagram on the one hand and a 1950'a vintage style electronic instrument on the other-these are the concepts of the block diagram and the front panel. LabVIEW is hierarchical in that any virtual instrument that we design (any complete functional unit it called a virtual instrument and is almost always referred to as a "VI") can be quickly convert in to a module which can be a sub-unit or another VI. This is entirely analogous to the concept of a procedure in a traditional procedure in traditional programming.

This language have two "faces":
- **the block diagram** - is almost the "back side" of the VI, practical is the program. It shows how the control and the indicators fit together as well licks the hidden modules where all the work gets done;
- **the front panel** – the face that the users sees practical is the user interface. It contains controls and indicators. By intelligent design of the front panel of a VI it is fairy simple to produce a simple clean design for the user.

We use this language to simulate two types of adders: carry select adder and carry completion adder.

2. CARRY SELECT ADDER

This type of adder is build use Ripple carry adder (RCA), who is almost the simple adder. We add two numbers represent at the four bits: a number $a = a_3a_2a_1a_0$ and a number $b = b_3b_2b_1b_0$, obtain the result $s = s_3s_2s_1s_0$, like can see in the front panel of the user interface, in figure 1.

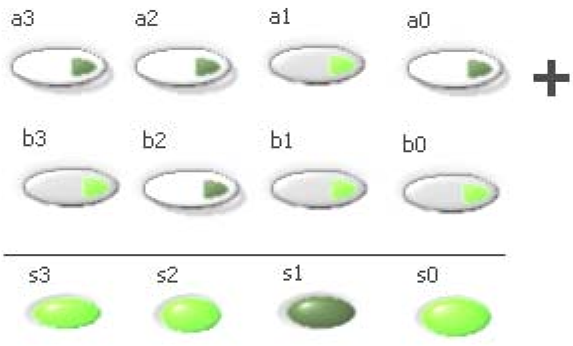


Fig. 1 The user interface

A button push represent 1 logic and a button who is not push represent 0 logic. Sow in this example we have:

0010 +
1011
1101

It is important to note that in LabVIEW a Boolean True/False is not the same as a numeric 1/0. We need to insert a *Boolean to (0/1)*. We need this in cases when to multiply a number by 0 or 1 dependent on the result of an operation. It probably would have been more correct to write the table using True and False, but it would have been too cluttered. We can also change numbers to Boolean operators. Careful when using this option because it takes the binary equivalent of the number, such as a decimal 6 is equal to 01100000000000000000000000000000 in binary. Before get angry, this is because it is represented as a 32 bit number, and written backwards because that is the way LabVIEW does it. We could change the Representation of 6 to a byte (again by right clicking) if we need to save memory. This can be helpful when controlling multiple switches and are in need of a control.

The block diagram, figure 2, represents the “back side” of this VI.

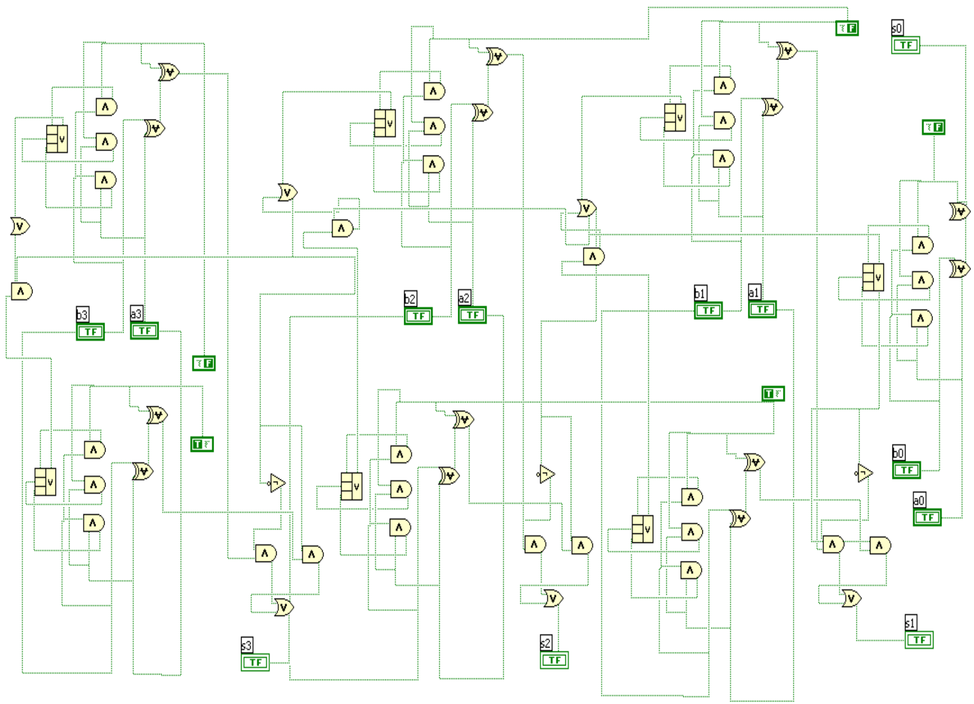


Fig. 2. The block diagram

This is the implementation of the carry select adder with logical gate and use the parallelization to obtain a better speed. In figure 4 it's present the principle diagram of this adder, where the RCA cells look like in figure 3. The equation use for RCA is:

$$\begin{cases} c_{i+1} = a_i b_i + a_i c_i + b_i c_i \\ s_i = a_i \oplus b_i \oplus c_i \end{cases}$$

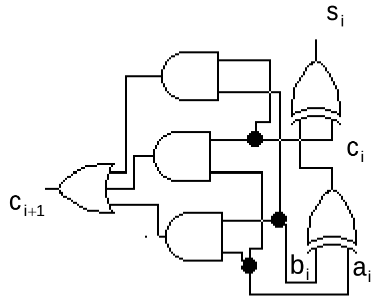


Fig. 3. Representation with logical gates of one cell

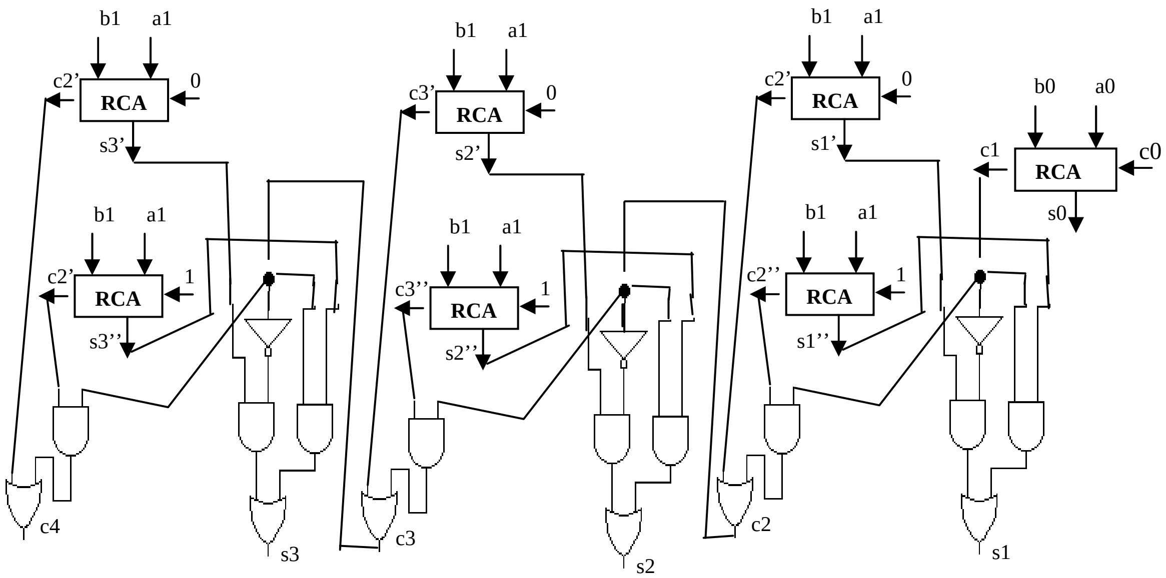


Fig. 4. The principle diagram

We don't expect for c_i and make a calculation of $a_i + b_i$ when carry is 0 and $a_i + b_i$ when carry is 1, that cos carry can be only 1 or 0. When carry c_i come from the precedent rang we make selection of the right carry for the next leval and a summe.

For example when add $a_1 + b_1$ in first cell, we consider carry 0 and make sum s1'and carry for the next leval c2'calculation. In secand cell we add $a_1 + b_1$, consider carry 1 and make sum s1'' and carry for the next leval c2''. When carry from the precedent rang c1 it come, we select s1' and c2' if it 0 logic and we select s1'' and c2'' if it 1 logic.

3. CARRY COMPLETION ADDER

This type of adder is build start with the idea to make two way to carry propagation (way 0 to propagation and way 1 to propagation). We add two numbers represent at the four bits: a number $a = a_3a_2a_1a_0$ and a number $b = b_3b_2b_1b_0$, obtain the result $s = s_3s_2s_1s_0$, like can see in the front panel of the user interface, in figure 5.

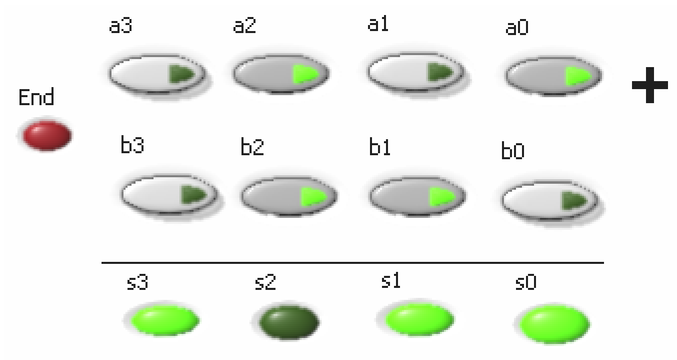


Fig. 5 The user interface

Imagine that we have just written a very complicated program. We feel really proud and run it, but it doesn't work quite right. When we look over the wire diagram we realize missed an important complicated calculation.

There is no way to fit it in the diagram, and if we tried, probably more mistakes would be created. Or we need a piece of code for several programs we are developing, and we don't feel like rewriting it every time. In text based programs, we could just type or copy the code in anywhere.

LabVIEW uses sub-Vies which are sub-routines for functions we may use many times. In fact most of the functions from file on down are sub-Vies. All sub-Vies are squares, but not all squares are sub-Vies. An easy way of determining this is by double clicking on the node. If a new VI opens, it is a sub-VI. Sub-Vies are completely self-sufficient programs, i.e. they can run on their own. The controls are the inputs and the indicators are the outputs.

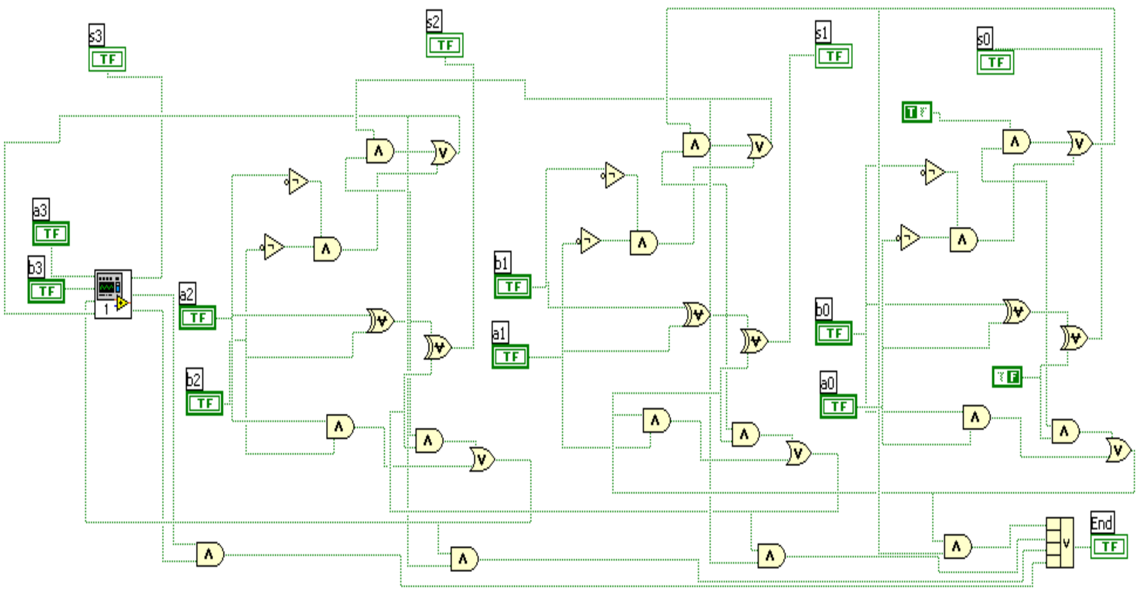
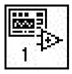


Fig. 6 The block diagram

How to make a sub-VI:

Write your subroutine with the controls as the inputs and the indicators as the outputs

Be sure to label the controls and indicators with descriptive and appropriate labels.

The little icon in the upper right hand corner , is where to make the node.

We use such a control to implement a cell of the carry completion adder, can see in figure 6, last part.

The block diagram, figure 6, represents the “back side” of this VI.

The equation use it is:

$$\begin{cases} c^0_{i+1} = \overline{a_i}\cdot\overline{b_i} + (a_i \oplus b_i)c^0_i \\ c^1_{i+1} = a_i b_i + (a_i \oplus b_i)c^1_i \end{cases}$$

determinate from the table bellow:

a_i	b_i	c_i	c_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

From this ecuations result the way to implement with logic gates one cell of this adder, figure 7.

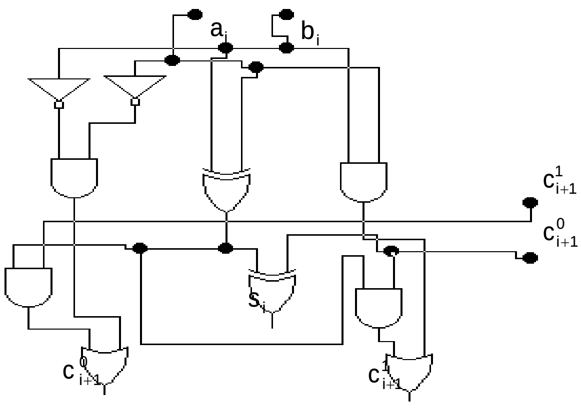


Fig. 7. Representation with logical gates of one cell

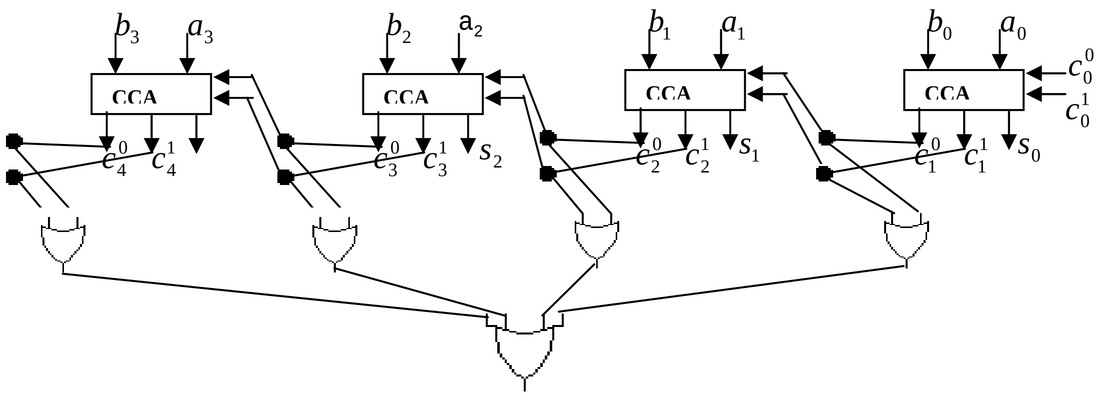


Fig. 8. The principle diagram

In figure 8 is present the principle diagram of this adder. The adder is end when the output of or logical gate it's 1 logic.

4. CONCLUSIONS

This easy aplications can be use to understand the basic knowledge of the adders, how to build a simulation using the LabIEW and how to make a sub VI.

LabVIEW it's used extensively in research and industry. To help in the design programs, the LabVIEW software provides an extensive library of functions and tools for data analysis, report generation, data acquistion and file input/output.

BIBLIOGRAPHY

[1.] Francis Cottet, Octavian Ciobanu, „Bazele programarii in LabVIEW". Matrix ROM, Bucuresti 1998;
[2.] http://www.ni.com/labview/;
[3.] http://c.webring.com/hub?ring=labview;
[4.] http://www.iit.edu/~labview/Dummies.html#FORNODE;
[5.] http://www.eelab.usyd.edu.au/labview/main.html;
[6.] http://www.mech.uwa.edu.au/jpt/tutorial/ieindex.html.