Unit 2

Digital systems

Labs of
Computer and Network Fundamentals
5th March 2014

SESSION 1

SECD: Educational Simulator of Digital Circuits

Objectives

The goal of this session is to introduce the SECD tool (SECD stands for *Simulador Educacional de Sistemas Digitales* or Educational Simulator of Digital Systems). This tool allows us to design a digital circuit in the computer screen, as well as to simulate its behaviour step by step, and to check the values of its outputs taking into account the values of its inputs.

Also, the input generator will be presented. This component facilitates the generation of values for the inputs of a digital circuit in order to test it.

Finally, the way in which new components based on basic circuits are built will be shown. These new components can be easily used in future projects.

Previous knowledge and required materials

In order to get the maximum benefit from this lab session, the student is required to:

- Understand the behaviour of a multiplexer.
- Understand the concept of truth table of a circuit, as well as to be able to write the truth table of a multiplexer.

Furthermore, it is convenient to go to the lab with a USB memory stick to store all the files created while working on the labs.

Session development

1. Building a simple digital circuit

SECD is a tool developed in Java and distributed as free software under GPL license. It is available from the web:

http://sourceforge.net/projects/secd

SECD requires the Java virtual machine installed in the computer, which can be downloaded from:

http://www.java.com/getjava/

Double click on SECD icon to run it. The main window of the program is shown in figure 1.1.

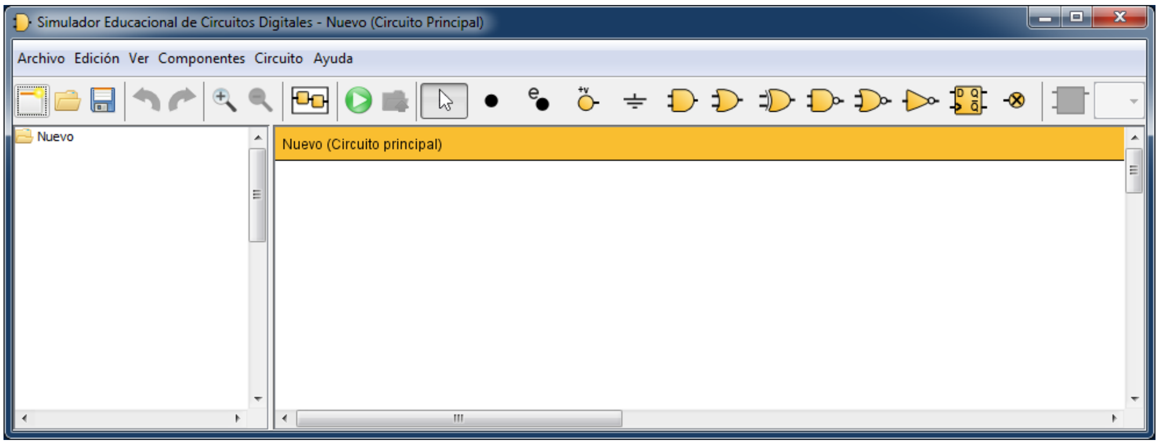

Simulador Educacional de Circuitos Digitales - Nuevo (Circuito Principal)
Archivo
Edición
Ver
Componentes
Circuito
Ayuda
Nuevo
Nuevo (Circuito principal)

Figure 1.1: Main window of SECD

Change the language to English by selecting the menu `View`→`Languages`.

The main window of SECD shows, from top to bottom, a menu bar with the simulation options, a toolbar with basic commands (create new project, open project, undo, redo, zoom, and select and insert components among others), a tree showing the files of the project, on the left, and the workspace, which is the area used to design the circuits. In this lab session you will learn to use all of these elements.

First, a multiplexer with two data lines (thus, with one select input) will be created. The final circuit is shown in figure 1.2.

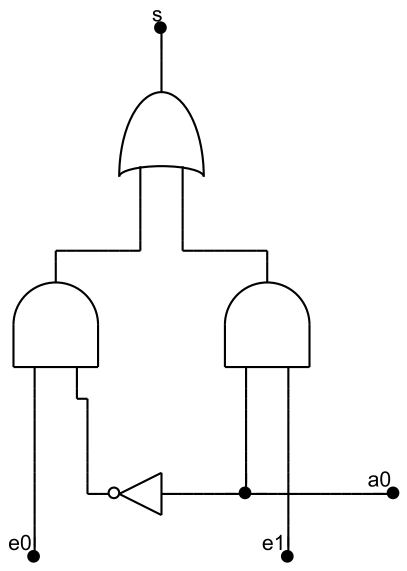


Figure 1.2: Diagram of a multiplexer with two data lines built up from logic gates

Select the AND gate in component toolbar and click in the workspace two draw two AND gates, as shown in figure 1.2. Next, place an OR gate as shown in the diagram of figure 1.2.

Now, select from the toolbar the NOT gate and place it under the two AND gates. As you can see, the gates are not oriented as they appear in figure 1.2. SECD allows the user to rotate the gates. In order to do this, you must choose the selection tool: click on the arrow in the toolbar, or right click in the workspace, or press Esc. The cursor changes to an arrow, showing that the selection tool has been chosen. Right click on the gate you want to rotate; the selected gate is highlighted in red. Then, you can rotate the gate by right clicking and choosing *Rotate* in the pop-up menu (you can use a shortcut: Ctrl-G, or the icons in the toolbar). In some occasions, you must rotate the gate several times to reach the desired orientation.

Next, a connection point (•) must be placed to the right of the NOT gate in order to link two wires, and connection points must be added to act as inputs. Select in the component toolbar the connection point (you can use the point that has en "e" to its left so that the program asks for the label) and place some of them as shown in figure 1.3.

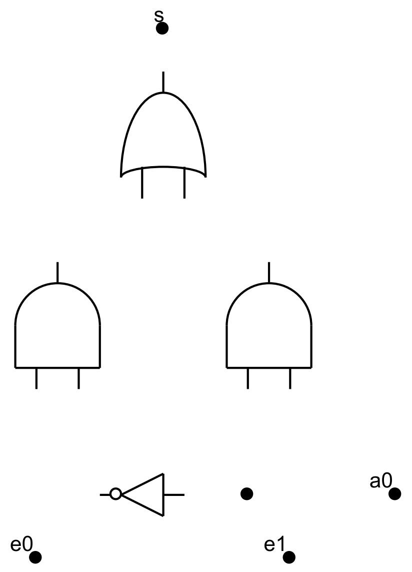


Figure 1.3: Connection points of a multiplexer with two data lines

As you can see, in this figure the inputs and the outputs are labeled. To label a point you must use the selection tool, and double click on the point. In order to simulate and to create new components, all the inputs need to be labeled.

Finally, you must connect the points and the gates using wires in order to implement the diagram of the multiplexer. To insert a wire, place the cursor at the pin of a gate. A black square will appear and the cursor will change to a hand (you can use the zoom to facilitate this action). Next, left click, do not release the button of the mouse, and drag the cursor to a connection point or a pin of another gate. Then, release the left button. The wire will appear, connecting the two elements.

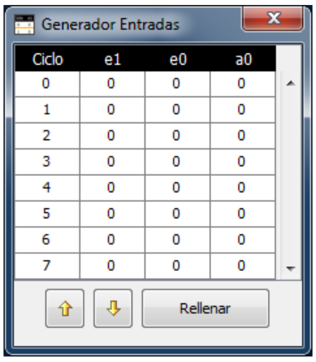
2. Verifying the circuit

In order to check whether the implemented circuit does really behave as a multiplexer, it will be necessary to introduce in its inputs all the combinations of logic values 0 and 1. Then, the output generated by the circuit must be compared with the truth table of a multiplexer. If they match, the implemented circuit is behaving like a multiplexer, so *it is* a multiplexer.

To examine the output of the circuit, connect a lamp to it. In the component toolbar, the lamp is represented as a cross rounded by a circle: $\otimes$. When the lamp is on (its color is green), the digital output is one. If it is off (its color is gray), the digital output is zero. The same color convention applies to the wires. If it is not known whether a component has a one or a zero value, its color is blue[1].

In order to generate different combinations for the inputs, a component called input generator will be used. This component appears when a circuit is being simulated and there are input connected to nothing but a wire. In the circuit, a connection point is considered an input if it is labeled and connected to only one wire.

Press the green button on the main toolbar. This button is used to simulate the circuit. When pressed, as there are inputs without any value, the input generator appears, as shown in figure 1.4.



Ciclo	e1	e0	a0
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0

Figure 1.4: Input generator

In the input generator, there are columns labeled with the label of each of the inputs of the circuit. You can write zero or one in each of the cells to write down the 8 possible combinations. However, the generator provides a button called *Fill*, which generates automatically all the possible combinations. Press this button and watch the results.

At this moment, the wires of the circuit are colored in blue, showing that they do not have any value. They are in an indefinite state. Press the button $\boxed{\downarrow}$ in the input generator and check that two things happen:

- The first row of the table of the input generator is highlighted in green. This shows that the input generator is currently generating the values of this row. That is, each of the inputs of the circuit contains the logic value of the column labeled with its name.
- The wires from all the inputs are colored in gray, meaning that their logic value is zero, which agrees with the values of the input generator in the highlighted row. These values will be propagated to the components of the circuit until reaching the lamp, which is actually gray, meaning that the output for the current values of the inputs is zero.

You can go forward or backward using $\boxed{\downarrow}$ and $\boxed{\uparrow}$ in the input generator, checking that all the combinations the output of the circuit agrees with the truth table of a multiplexer. How many input combinations generate a one in the output?$\boxed{1}$

1

[1]The default colors can be changed selecting the option *View→Configure simulation colors* of the menu bar.

Now, you are going to store the circuit. First, you must stop the simulation pressing the "Stop" button in the toolbar. Then, press the button with the disc icon. A dialog box will appear asking for the folder to save the project. Choose the folder, insert `mux2` as the name for the project and press *Save*. In order to check that the project has been saved correctly, open the Windows explorer and navigate to the folder in which you saved the project. In this folder, there should be one file: `mux2.pro`. It is the project file. A project is a main circuit that can use new components created by the user. Later, you will create these new components and this file will store information about these new components.

3. Creating user-defined components

In the previous section, a two-data-input multiplexer was designed, implemented, and checked. Now, a new component based on this circuit will be created. Then, when a two-data-input multiplexer is required, it will not be necessary to implement the multiplexer again. Instead, the new component will be used.

The procedure to create a new component, also called macro, requires that the connection points that will be inputs and outputs of the new component are not connected to other components but only to a wire. Therefore, it is necessary to remove the lamp connected to the output of the circuit: select the lamp and press Supr.

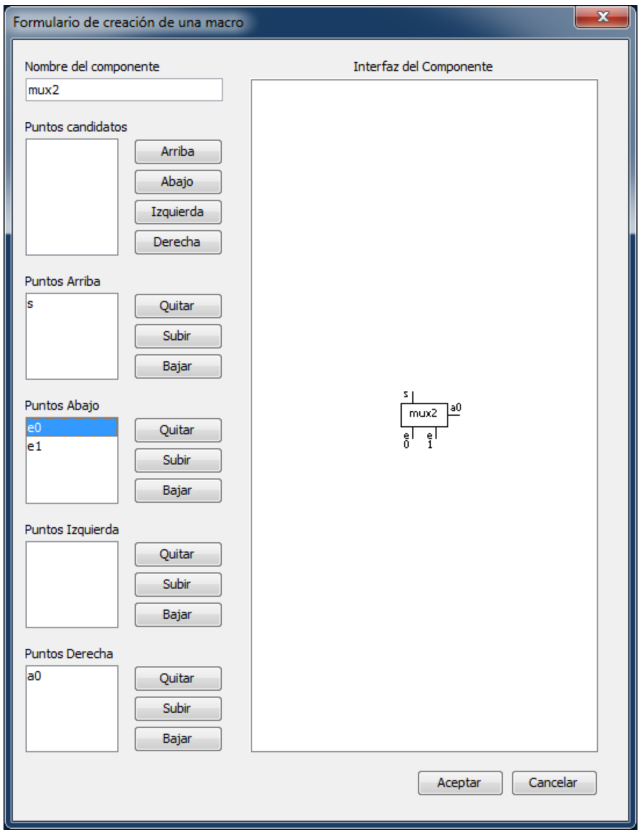
Once the circuit is prepared, the procedure to create a new component consists of the following steps:

- ❏ Press the create component button in the toolbar. This button displays a white rectangle with two orange rectangles inside.
- ❏ The new macro dialog box will appear. The component will be called `mux2`, taking the name from the project name, but you could change it. In this case, we want to call the component `mux2`, so it should not be changed.
- ❏ In the dialog box you can see several lists. The first one contains the candidate points to be inputs or outputs before a position in the sketch of the new component is assigned. The following lists contain the connection points which will appear at the top, at the bottom, on the left, and on the right of the interface of the new component. Using the buttons located to the right of the first list, create the macro with the appearance of figure 1.5. Notice that `e0` must be on the left of `e1`.
- ❏ Press *OK* to create the new component. You should notice the new component in the toolbar, in the combo-box at the top-right of the screen.

Save the circuit at this moment. Using the Windows explorer, go to the folder of the project. You will see a new file: `mux2.cir`, which contains the new component you just created.

4. Simulating user-defined components

Now, a new circuit will be created from the current project to understand how user-defined components work.



Nombre del componente: mux2

Puntos candidatos:

Puntos Arriba: s

Puntos Abajo: e0, e1

Puntos Izquierda:

Puntos Derecha: a0

Interfaz del Componente

Figure 1.5: Creating the macro `mux2`

- ❏ Erase the circuit drawn in the main circuit (in the workspace). To do so, use the selection tool: draw a rectangle to select all the components of the circuit. Then, press Supr.
- ❏ In the toolbar, select the `mux2` component. To do so, you must press the icon closer to its name. The cursor will change to indicate that you have selected the component.
- ❏ Place the component in the workspace.
- ❏ Now you should insert connection points for the inputs and outputs of the circuit and then you should label them as shown in figure 1.6. Adding connection points to components and labelling these connections points is a very common operation, so SECD provides a function for this: right-click the component and choose the option *Place points*. Connect also a lamp to the output of the circuit, as shown in this figure.
- ❏ Save the project. The main circuit will also be saved.
- ❏ Simulate the circuit by pressing the green button (the simulation button). The input generator will appear. Press the button *Fill* to generate all the possible combinations for the inputs. Then, using the button ↓, check that the output behaves as expected.
- ❏ Go to the row associated with the cycle number 3. Choose the selection tool in the component bar in the main window, if it was not already chosen. Then, double click on `mux2` in the workspace. You will go into the circuit `mux2` and you will see the values for the inputs, outputs, and wires. This is a very useful piece of information to understand complex circuits built from simpler circuits.

❏ To go back to the main circuit, press the button with the yellow arrow pointing to the left in the toolbar. Notice that the main circuit of a project shows an orange frame at the top of the window with the name of the project and the label *Main circuit*. When the workspace is showing the interior of a component, the label shown is the name of the component followed by *Defined component* and the frame and the background change to grey. This piece of information will be very useful for you to avoid making mistakes while creating user-defined components, which should be created only in the main circuit of the project. Be careful while working with circuits files. If you rename them, they, as well as the components which they store inside, could behave unexpectedly. Take into account that all the components for a project must be located in the same folder on disk.

❏ To navigate in the files of a project (the main file, which corresponds with the project, and the components associated to it) it is also possible to use the navigation tree shown on the left of the main window of SECD. However, if you try it now, you will get an error message because while the simulation is running, there can be several instances of the same component with different states and SECD can not know from the tree which one you want to open. Thus, before opening a component from the tree, you must stop the simulation. Stop it now and then click on `mux2`: you will see that the component file will be open.

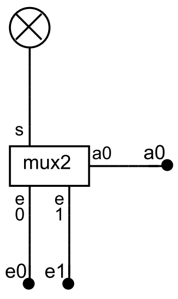


Figure 1.6: Testing the user-defined component `mux2`

5. Creating a 4-input data multiplexer

A 4-input data multiplexer, seen as a black box, has 6 inputs (4 data inputs and 2 select inputs) and 1 output. The data inputs will be labeled `e0`, `e1`, `e2`, and `e3`, and the select inputs `a0` and `a1` (the number shows the weight of the bit). The output will be labeled `s`.

There are two approaches to create a 4-input multiplexer:

- Write the truth table, and deduce its logic function from it.
- Take advantage of the `mux2` component developed in the previous session.

The first approach is direct and automatic, but it presents a drawback: it is extremely tedious in this case since the truth table of a 4-input data multiplexer is huge (it would consist of $2^6 = 64$ input combinations). This approach is recommended for the conscientious student.

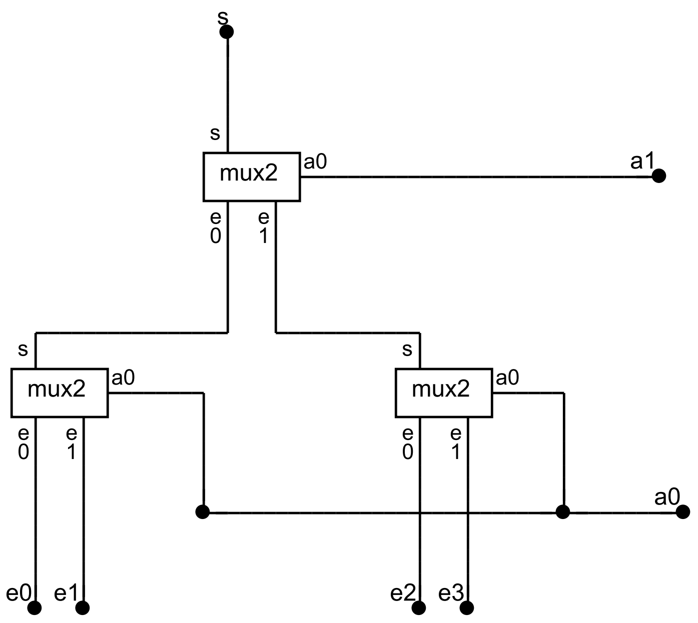


Figure 1.7: Three 2-input data multiplexer to create a 4-input data multiplexer

In this lab session the second approach will be used. To do so, three 2-input data multiplexers are needed (that is, three instances of the macro `mux2`). In figure 1.7 the way in which the three macros are connected is shown. You must follow these steps:

- ❏ Open the project `mux2` if you do not have it already loaded.
- ❏ Save the project as `mux4` and remove the elements from the main circuit (be sure you are in the main circuit in SECD).
- ❏ Create a circuit following the diagram shown in figure 1.7, using the component `mux2`.
- ❏ Create a macro from the circuit, and label it as `mux4`. Configure the inputs and outputs as shown in figure 1.8.

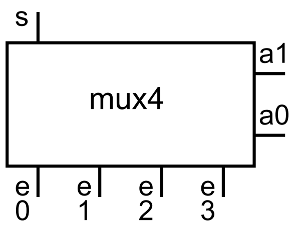


Figure 1.8: Interface of a 4-input data multiplexer

Next, a new circuit will be created to check the macro.

- ❏ Delete the circuit you have in the main circuit. To do so, use the selection tool by drawing a rectangle over all the components of the circuit. Then, press Supr.
- ❏ In the toolbar, select the component `mux4`. To do so, you must press on the icon next to its name.

❏ Place the component in the workspace.

❏ Place connection points for the inputs and output and label them with the command *Place points*. Place also a lamp connected to the output.

❏ Save the project. The main circuit will also be saved.

Next, and before the simulation process, fill the following table based on what you expect to appear in the output `s` for each of the input combinations.

e3	e2	e1	e0	a1	a0	s
0	0	0	0	0	1	
0	0	0	0	1	1	
0	0	1	0	1	0	
0	1	0	0	1	0	
0	1	1	1	1	1	
1	1	1	1	0	0	

Once the table has been filled, press the simulation button. The input generator will appear. Insert the combinations for the inputs shown in the table and check your answers.

Bear in mind that while simulating the circuit, if you double click on any component, you can see its internal state.

6. Supplementary exercises

6.1 Files in your working folder

In your working folder you must have the following files:

- `mux2.pro` and `mux2.cir`: the project with the main circuit, and the circuit of the component `mux2`, respectively.
- `mux4.pro` and `mux4.cir`: the project with the main circuit, and the circuit of the component `mux4`, respectively.

6.2 Exercises

✏ Open the `mux2` project. Press the simulation button and write in the first row of the input generator the following values: `e0=1`, `e1=0`, and `a0=0`. If you pressed the button $\boxed{\downarrow}$, what color will show the lamp?[2]

2

✏ Create a new project (*File→New project*) that implements a NAND logic gate with four inputs. From this circuit, create a user defined component.

✏ Create a new project. Then build the circuit of the figure 1.9, which it has three inputs and one output.

Connect the output of the circuit to a lamp. Before simulating the circuit, fill the following truth table:

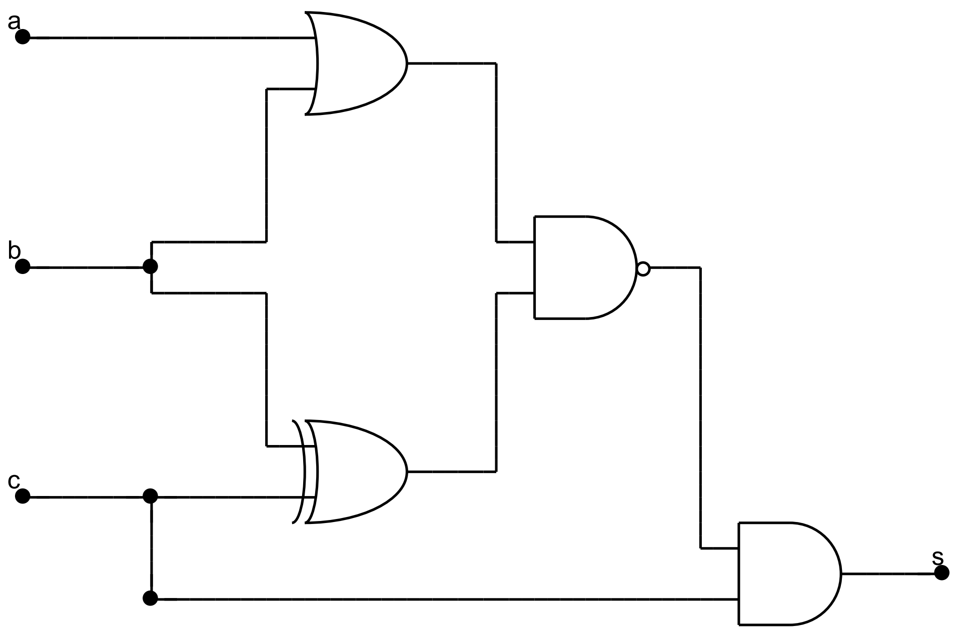


Figure 1.9: Circuit for suplementary exercises

a	*b*	*c*	*s*
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Simulate the circuit. Check that the circuit behaves as you predicted in the above truth table.

✏ Create a macro which implements the following logic function: $s = a \cdot (\overline{b + c})$. Then, check that it works correctly.

SESSION 2

Building an ALU

Objectives

The aim of this session is to create and simulate the ALU of the CT (*Computador Teórico*). The CT is a 16-bit computer which will be used in the course to study the main concepts about computer fundamentals. This ALU will be able of performing arithmetic operations, such as addition and subtraction, as well as logic operations, such as AND, OR, and XOR.

In order not to make the simulation process complicated, the ALU will operate with 4 bits, but its operation will be the same as the 16-bit ALU of the CT.

Previous knowledge and required materials

- Unit 2 of the course book, *Fundamentos de Computadores y Redes*, specifically the section 2.2.7, where the design and development of the ALU is covered. The student must read carefully this section before attending the lab session to understand the internal behaviour of the ALU of the CT.
- Project `mux4`, developed in the previous session. This project contains components that will be required to create the ALU.

Session development

1. Introduction

This lab session is intended for the student to create a 4-bit ALU similar to the 16-bit ALU used in the CT. To do so, a 1-bit ALU will be created firstly. Then, using four 1-bit ALUs, an overflow detector, and additional logic, the 4-bit ALU will be created. Finally, the ALU will be tested carrying out several arithmetic and logic operations.

2. Creating an ALU

Creating an ALU requires the following tasks:

- ❏ Download the project `alu1` from the Campus Virtual. This project has some points and connections required to build the 1-bit ALU.
- ❏ Start `SECD` as usually and load the `alu1` project.
- ❏ To create the 1-bit ALU we need a component that we have not created: an 1-bit adder. Download from the Campus Virtual the file `sum1.cir`, which contains the description of this component, and use in SECD the option `File`→`Add existing file` to add it to the project.
- ❏ We will also need the components `mux2` and `mux4` created in the previous session. Add `mux4` with the option `Add existing file`. As `mux4` uses `mux2`, the latter will be automatically included in the project.
- ❏ Create the circuit of the 1-bit ALU. You can use the figure 2.1 as a guide. Save the project.

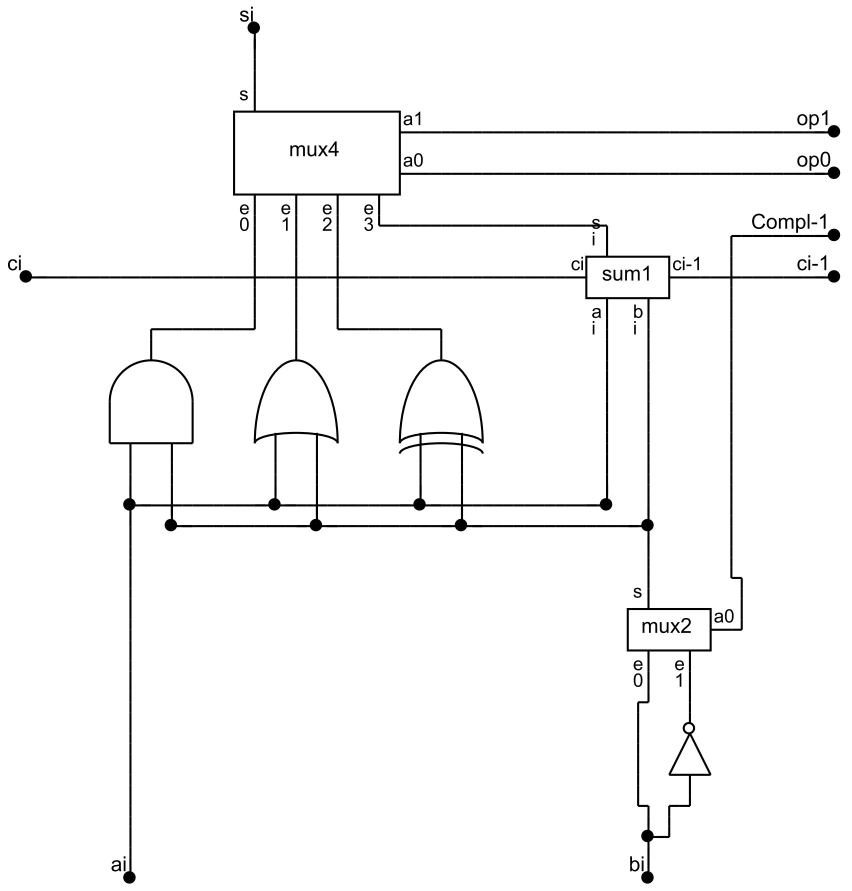


Figure 2.1: 1-bit ALU internal scheme

- ❏ Create the macro `alu1`, with inputs and outputs located as shown in figure 2.2.
- ❏ Check again that the inputs and outputs of the macro match the scheme shown in figure 2.2. It is a common mistake to swap the location of the inputs `Compl-1` and `ci-1`. Another common error is putting `bi` to the left of `ai`.

Now, we will start creating the 4-bit ALU. We will carry out its development in two steps. Firstly, we will design an ALU without flags. Then, starting from this ALU, we will create the full ALU. To do so, follow the next steps:

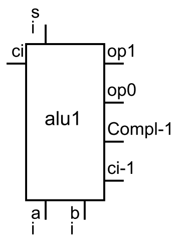


Figure 2.2: Macro of a 1-bit ALU

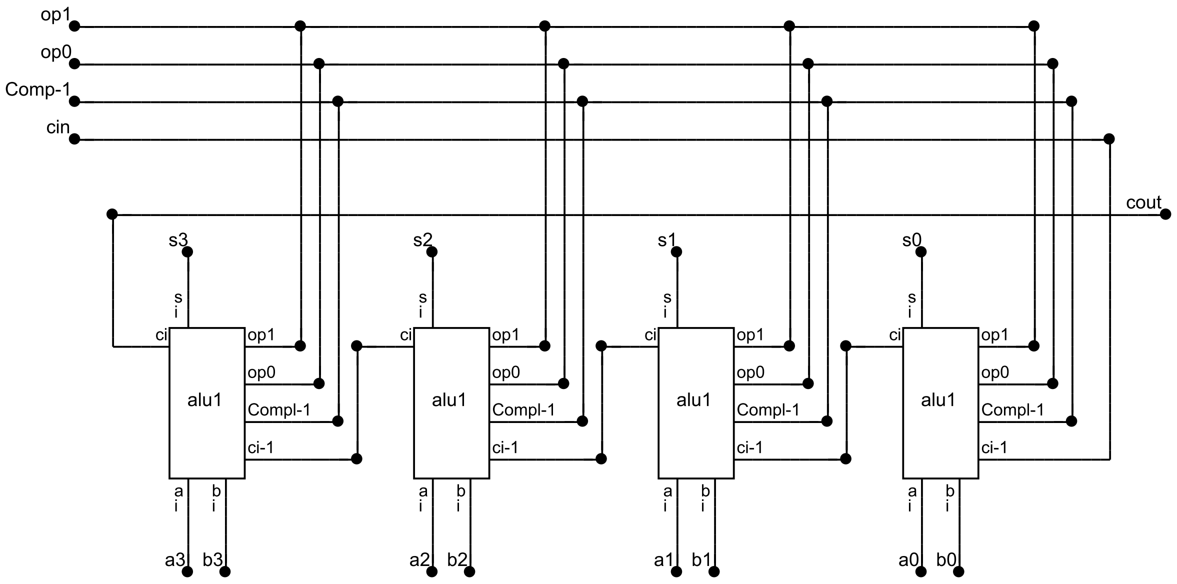


Figure 2.3: Internal scheme of the 4-bit ALU without flags

- ❏ Download from the Campus Virtual the file `alu4-nf.pro` to your working folder and open it in SECD. This project contains the basic points and cables to create the 4-bit ALU with no flags.
- ❏ Use in SECD the option `File`→`Add existing file` to add the component ALU1 to the project. When you do this, all the components used in ALU1 will be also added to the project recursively.
- ❏ Build the 4-bit ALU without flags. You can use figure 2.3 as a design guide in order to place and connect the macros. If SECD shows an error when trying to connect two distant points, the best solution is to add intermediate points to facilitate the software solving the routing problem for the wires.
- ❏ Once the circuit has been built, create the `alu4-nf` macro, placing its inputs and outputs located as shown in figure 2.4. Notice again that inputs `ai` must be to the left of inputs `bi`. Save the project.

Finally, we will create the full ALU, with flags, following the next steps:

- ❏ Save the project as `alu4`.
- ❏ Remove the elements of the main circuit and save the project again.
- ❏ To obtain the flags you will need a circuit to compute the overflow for the addition. Download from the Campus Virtual the file `gen_ov_s.cir` and add it to the project with the option `File`→`Add existing file`.

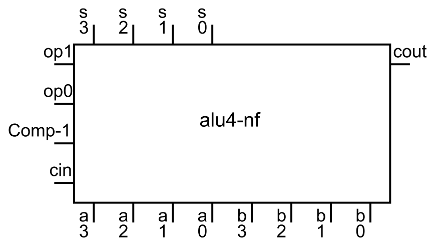


Figure 2.4: Macro of a 4-bit ALU without flags

- ❏ You will also need a NOR logic gate with four inputs to generate the Z flag. Download from the Campus Virtual the file `nor4.cir` and add it to the project with the option `File`→`Add existing file`.
- ❏ Build the circuit of the 4-bit ALU with flags. To do so, you must start from the `alu4-nf` macro. **Notice that, for considering a point an input or an output, it must have a label and exclusively one cable connected to it.**
- ❏ Once the circuit has been built, create the `alu4` macro, with its inputs and outputs located as shown in figure 2.5.

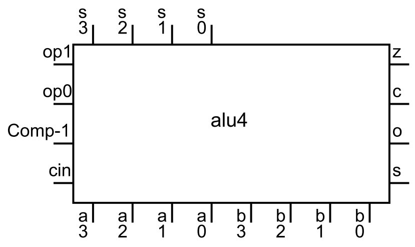


Figure 2.5: Macro of a 4-bit ALU

3. Testing the ALU

In this section, several arithmetic and logic operations will be simulated on the ALU to test its functionality.

- ❏ Fill the next table making the operations by paper and pencil, that is, without using SECD.

A	B	Operation	Result	z	c	o	s
0110	0111	Add					
0110	0111	Sub					
0110	0111	OR					
0110	0111	AND					
0110	0111	XOR					

- ❏ Remove the elements of the main circuit of the workspace.
- ❏ Select the component `alu4` in the tool bar and place it in the workspace.
- ❏ Create connection points for inputs and outputs and label them using the command *Place points*. Also, connect a lamp to each output.

❏ Press the simulation button; the input generator will show up. Write the required values for the inputs to perform the operations of the above table.

❏ Verify that your answers are correct comparing them with the results provided by SECD.

4. Supplementary exercises

4.1 Files in your working folder

In your working folder you must have the following files:

- The files created in the previous session and the files `sum1.cir`, `gen-ov-s.cir` and `nor4.cir` that you downloaded from the Campus Virtual.
- `alu1.pro` and `alu1.cir`: the project with the main circuit, and the circuit of the component `alu1`.
- `alu4-nf.pro` and `alu4-nf.cir`: the project with the main circuit, and the circuit of the component `alu4-nf`.
- `alu4.pro` and `alu4.cir`: the project with the main circuit, and the circuit of the component `alu4`.

4.2 Exercises

✏ Fill the next table, making the operations by paper and pencil, that is, without using SECD.

A	B	Operation	Result	z	c	o	s
0010	1100	Add					
0010	1100	Sub					
0010	1100	OR					
0010	1100	AND					
0010	1100	XOR					

Verify that your answers are correct comparing them with the results provided by SECD.

✏ Think of two operands whose addition generates carry. Perform the addition by paper and pencil, and calculate the flags. Then, simulate the operation in SECD and check your results.

✏ Think of two positive integer operands whose addition generates carry. Do these operands exist? Why?

✏ Think of two negative integer operands whose addition generates carry. Do these operands exist? Why?

- ✏ Think of two positive integer operands whose subtraction provides a negative result. Calculate the result of the subtraction and the flags. Then, simulate the operation in SECD and check your results.
- ✏ Think of two positive integer operands whose subtraction generates overflow. Do these operands exist? Why?

SESSION 3

Integrating the ALU in the CT

Objectives

The main goal of this lab session is for the student to simulate the arithmetic and logic operations inside the CPU of the CT (*Computador Teórico*). In order to simplify the simulation, we will consider the CPU as a 4-bit CPU, but analogue to the CPU of the CT. Furthermore, only the next elements of the CPU will be considered:

- Internal bus.
- ALU.
- Temporal input register.
- Temporal output register.
- Flags.

After carrying out this lab session, the student should be able to describe the required sequence of signals to perform an arithmetic or a logic operation in the CPU of the CT.

Previous knowledge and required materials

- Understand the operation of both a register and the ALU of the CT.
- Project `alu4`, developed by the student in the previous lab session. This project contains, among others, the `alu4` macro.

Session development

1. Introduction

In this section, the ALU will be connected with the temporal input register, with the temporal output register, and with the internal bus. The objective is to perform several arithmetic and logic operations. In order to facilitate the development of this circuit, the project `cpu` is provided.

2. Developing the circuit

The next steps must be followed:

- ❏ Download from the Campus Virtual the file `cpu.pro` to your working folder containing the rest of the files created in the previous sessions of this lab unit.
- ❏ Start SECD as usually.
- ❏ Create a new project, and save it with the name `reg4`.
- ❏ Develop the circuit of a 4-bit register following the design shown in figure 3.1. Save the project.

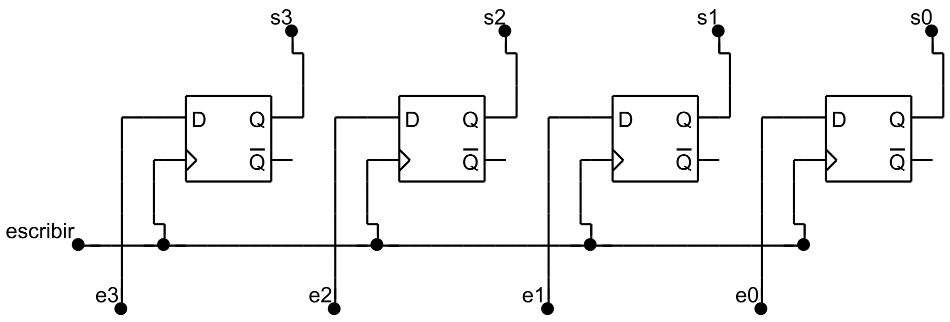


Figure 3.1: Internal scheme of a 4-bit register

- ❏ Create, from the circuit, the macro `reg4`. Place its inputs and outputs as shown in figure 3.2.

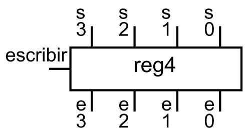


Figure 3.2: Macro of a 4-bit register

- ❏ Load the project `cpu`. The workspace of SECD will show the circuit of figure 3.3. ALU, TMPE, TMPS, and SR stand for the places where the macros `alu4` and `reg4` will be connected. Furthermore, four sets of lamps were located in interesting locations of the circuit:
  - The internal bus (the four vertical wires on the right).
  - Output of the status register.
  - Output of the temporal output register.

  To watch the value of the rest of the elements of the circuit while simulating, you must observe the color of the wires.
- ❏ Add the file `alu4.cir` using the menu `File`→`Add existing file`. The tool will tell you that all the components used by `alu4` have also been added.
- ❏ Following the same procedure, add the component `reg4`. Save the project.
- ❏ Place on the workspace, in the location specified by the label ALU, an instance of the macro `alu4`.

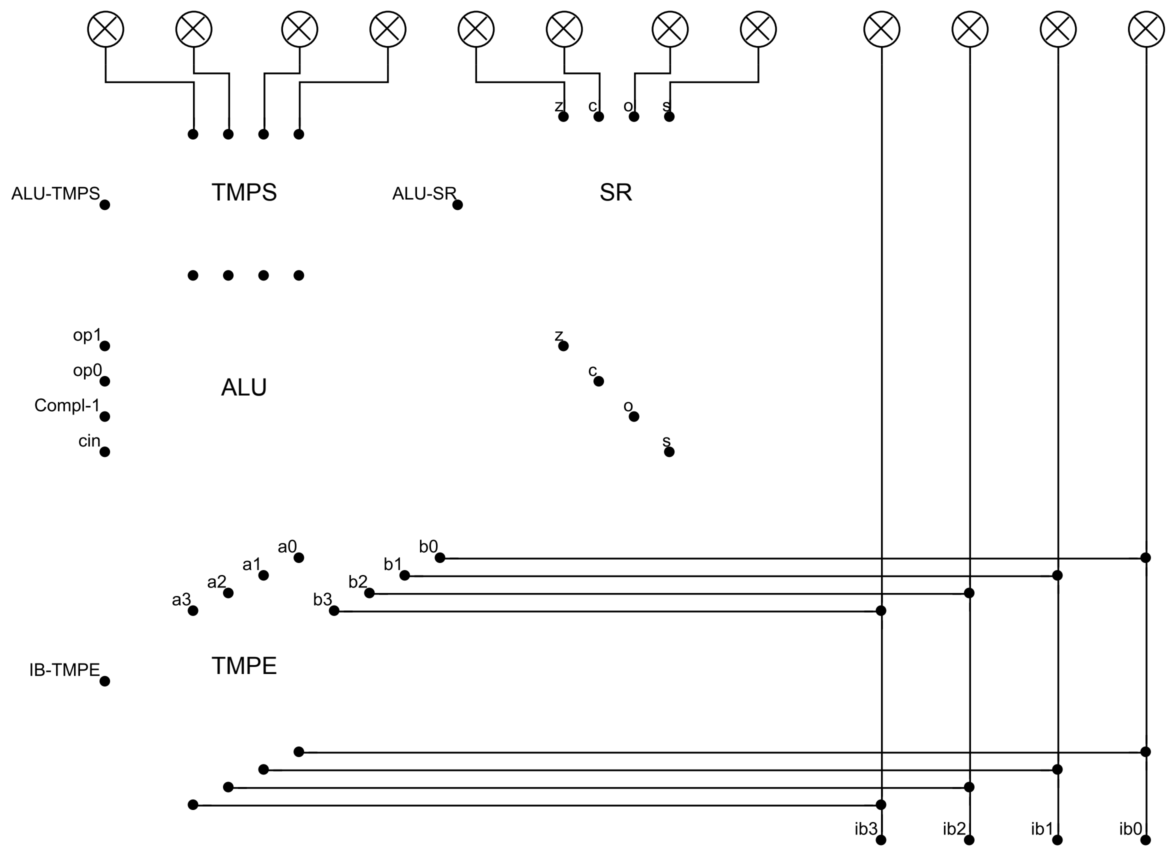


Figure 3.3: CPU circuit

❏ Place on the workspace, in the location specified by the label TMPE, an instance of the macro `reg4`.

❏ Place on the workspace, in the location specified by the label TMPS, an instance of the macro `reg4`.

❏ Place on the workspace, in the location specified by the label SR, an instance of the macro `reg4`.

❏ Make the appropriate connections among the macros and the connection points of the circuit. Once at this point, the circuit implements the scheme shown in figure 3.4.

❏ Perform, using paper and pencil, the subtraction of the operands $A = 5$ and $B = 4$, as shown in figure 3.5.

❏ Simulate the subtraction of the operands $A = 5$ and $B = 4$. To do so, you must follow the next steps:

1. Press the simulation button. The input generator will show up. A column for the next inputs of the circuit should appear: `op1`, `op0`, `ib3`, `ib2`, `ib1`, `ib0`, `IB-TMPE`, `Compl-1`, `cin`, `ALU-TMPS` y `ALU-SR`. If any of these inputs is missing, or any other than these appears, you must revise your circuit before continuing with the next step: you made a mistake while creating the circuit.
2. Place the first operand in the internal bus. That is, write the pattern `0101` in the 0-cycle of the input generator. Each bit must be located in the column `ib3`, `ib2`, `ib1`, and `ib0`, respectively.

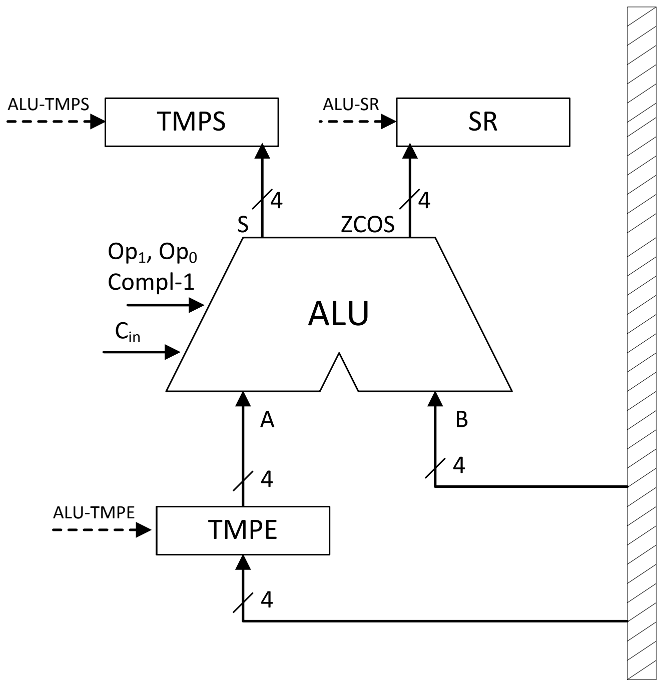


Figure 3.4: Block diagram of the CPU

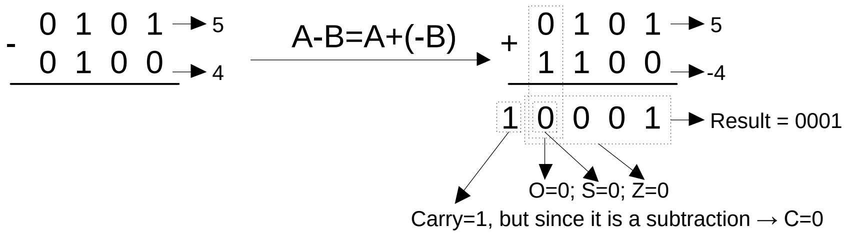


Figure 3.5: 5-minus-4 operation

3. Press ↓ in the input generator to pass the bit pattern of the 0-cycle row to the circuit. As you can see, the internal bus contains the pattern `0101`, and this pattern reaches the input of TMPE register. However, the output of this register is undefined because no pattern was written in it.
4. Now, the bit pattern in the internal bus will be written in the TMPE register. To do so, write a `1` in the 1-cycle row of the input generator in the column `IB-TMPE`. As you can see, this input, or signal, is connected to the `escribir` (`write`) signal of the TMPE register.
5. Press ↓ in the input generator to pass the bit pattern of the 1-cycle row to the circuit. Watch how the bit pattern at the output of the register has changed: it is the same pattern as in the internal bus when the rising edge of the `IB-TMPE` signal occurred (the rising edge is the signal transition from `0` to `1`).
6. Place the second operand in the internal bus. That is, the bit pattern `0100`. Furthermore, activate the following control signals of the ALU: `cin=1`, `Compl-1=1`, `op0=1`, and `op1=1`. These values must be loaded in the cycle 2.
7. Press ↓ in the input generator to pass the bit pattern of the 2-cycle row to the circuit. At this moment, the output of the ALU shows the

result, as well as the ZCOS bits. Check if they are correct. If they are not, you made a mistake in the previous steps. If the inputs of the ALU are correct, but the result is incorrect, you made a mistake while developing the ALU. Open its circuit and check each value to find the mistake.

8. Finally, the result will be written in the output temporal register, and the flags in the status register. To do so, write a `1`, in the 3-cycle row, in the signals `ALU-TMPS` and `ALU-SR`.
9. Press $\boxed{\downarrow}$ in the input generator. You will see how the result and the flags are stored in TMPS and SR, respectively.

❑ Check that the result and flags obtained in SECD match the result and flags obtained by paper and pencil.

❑ Save the project.

3. Supplementary exercises

In these lab sessions, where digital systems are simulated, it is very important for you to predict the results before performing the simulation process. In the following exercises, you must compare your prediction with the results of the simulation provided by SECD, and draw your own conclusions. If your prediction does not match the result of the simulation, it may be possible that you need to revise the main concepts involving digital systems. Or, it may be possible that the implementation of the simulated circuit were incorrect. In any case, you must invest time until your predictions match the simulation results.

In particular, in this lab session you must predict the result and flags of any arithmetic and logic operation before simulating the circuit. Furthermore, you must predict the value of any wire of the circuit during the simulation process.

3.1 Files in your working folder

In your working folder you must have the following files:

- `reg4.pro` and `reg4.cir`: the project with the main circuit, and the component `reg4`.
- `cpu.pro`: the project with the main circuit of the CPU.

3.2 Exercises

✏ Perform the arithmetic and logic operations of the following table. Firstly, by paper and pencil,[1] and secondly by simulating them in SECD.

[1] In logic operations it is enough to compute the Z bit, due to the rest is irrelevant.

Operand A	Operand B	Operation	Result	ZCOS
-3	7	OR		
5	3	Add		
-6	-8	Sub		
4	-6	Add		
-3	7	Sub		
6	-2	AND		
7	3	Sub		
-1	-5	XOR		
5	-4	AND		

✏ As you have seen in this session, an arithmetic or a logic operation requires 4 cycles. However, an option to reduce the number of cycles is writing in the registers at the middle of a clock cycle instead of at the beginning. Using this technique, how many cycles would it be necessary to perform an arithmetic or a logic operation?