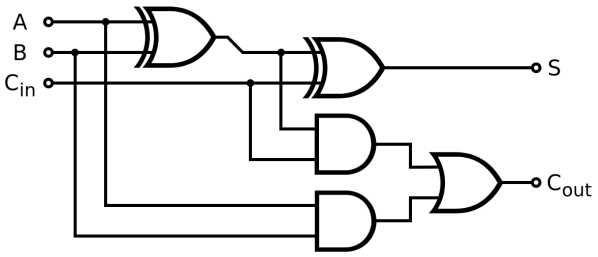
Netlist specification

Georges-Axel Jaloyan <georges-axel.jaloyan@ens.fr>
Balthazar Patiachvili <balthazar.patiachvili@ens.psl.eu>

1 Introduction to netlists

A netlist is the description of an electronic circuit. This description contains the logic gates that compose the electronic circuit, along with its inputs and outputs.
In the netlist language, a *full-adder* can be described like this:

```
INPUT a, b, c_in
OUTPUT s, c_out
VAR
_l_10, _l_9, a, b,
c_in, c_out, s, t
IN
c_out = OR _l_9 _l_10
s = XOR t c_in
t = XOR a b
_l_9 = AND a b
_l_10 = AND t c_in
```



We can notice a few things.
First, although we are only interested in the inputs (`a`, `b`, `c_in`) and the outputs (`s`, `c_out`), the netlist contains several intermediate variables (`t`, `_l_9`, `_l_10`).
Second, the netlist is not ordered: for example, the computation of `s` depends on `t` whose computation is described after (on the line after).
Third, each equation correspond to a logic gate in the circuit: this circuit contains 5 equations therefore 5 logic gates.

2 The netlist langage

2.1 Concrete syntax

A netlist has the following layout:

```
INPUT [inputs]
OUTPUT [outputs]
VAR [vars] IN
  [equations]
```

`[inputs]` and `[outputs]` are the inputs and outputs of the circuit, given as a comma-separated identifier list.
`[vars]` gives the type of all inputs, outputs and local variables, given as a comma-separated list of either:

- `x` representing a 1-bit variable,
- `x:n` representing a bus with n bits.

Finally, `[equations]` represents the circuit logic gates.

2.2 Equations

2.2.1 Boolean operations

- `x = constant` (in binary),
- `x = NOT a`,
- `x = AND a b`,
- `x = OR a b`,
- `x = NAND a b`,
- `x = XOR a b`,
- `x = MUX choice a b`, meaning that `x` is equal to `a` if `choice` is false, otherwise it is equal to `b`.

2.2.2 Bus operations

To help improve the cable management, wires can be grouped together in buses.

- `x = CONCAT a b`, representing the concatenation of the two buses `a` and `b`,
- `x = SELECT i a`, representing the selection of the `i`th bit in the bus `a`,
- `x = SLICE i1 i2 a`, representing a slice from the `i1`th bit to the `i2`th bit (inclusive) from the bus `a`,

2.2.3 Delaying a value (register)

Even though it could theoretically be possible to design registers using the previously described operations (e.g. by creating a flip-flops), they introduce combinatorial cycles that hinder the simulation of the circuit. For this purpose, the following operation isolates those constructs into a black box.

- `x = REG a`, meaning that `x` is equal to the previous value of `a`.

2.2.4 Volatile and non-volatile memory (RAM and ROM)

Similarly, RAMs and ROMs could be designed using previously described operations. For several reasons (exercise: guess them!), we still provide operations to do them in a single equation.

- `x = ROM addr_size word_size read_addr`,
- `x = RAM addr_size word_size read_addr write_enable write_addr write_data`.

The parameters for `ROM` and `RAM` are the following:

- `addr_size`: the number of bits needed to represent an address (an integer)
- `word_size`: the number of bits that you read at once (an integer)
- `read_addr`: the read address (a variable with bus size `addr_size`)
- `write_enable`: a switch to enable writing to memory
- `write_addr`: the write address (a variable with bus size `addr_size`)
- `write_data`: the data to write in memory (a variable with bus size `word_size`)

Note that RAM can work like registers: it can read and write at the same address during the same cycle without introducing a combinational cycle. Indeed, the read value is the one from the previous cycle. Hence, the written value can appear in the cycle after, but not in the current cycle.

2.3 Netlist scheduling

To simulate the netlist, it may be useful to start by ordering it, so that equations can be simulated top-down.
Below is a short unordered netlist example for a full-adder circuit, and its ordered version on the right, as seen by your simulator:

```
INPUT a, b, c_in
OUTPUT s, c_out
VAR
_l_10, _l_9, a, b,
c_in, c_out, s, t
IN
c_out = OR _l_9 _l_10
s = XOR t c_in
t = XOR a b
_l_9 = AND a b
_l_10 = AND t c_in
```

```
INPUT a, b, c_in
OUTPUT s, c_out
VAR
_l_10, _l_9, a, b,
c_in, c_out, s, t
IN
t = XOR a b
_l_9 = AND a b
_l_10 = AND t c_in
s = XOR t c_in
c_out = OR _l_9 _l_10
```

Note that in the version on the right, the inputs of each equation are either inputs of the netlist or outputs of previous equations. This property can be used to simulate the netlist top-down.

3 Writing netlists

Writing netlists by hand is tedious! You can use an HDL (Hardware Description Language) to make your life easier.
By default, we recommend using `carotte.py`, an HDL we designed for this course that is shallowly embedded in Python. It is available at `https://www.github.com/TWal/carotte.py`. Did you find bugs in `carotte.py`? Please open an issue or a pull-request!
Other options include using MiniJazz, a legacy HDL used previously in this course, or designing your own HDL in your favorite language! However, make sure you are able to generate all the example netlists that you were given in TP1.