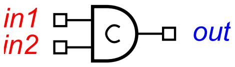
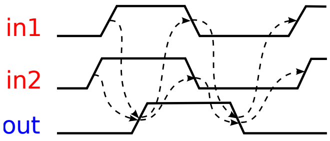
Workcraft

Synthesis and verification of C-element

C-element [http://en.wikipedia.org/wiki/C-element] is a latch that synchronises the phases of its inputs. A symbol for a 2-input C-element and its timing diagram are shown in the figures below. Initially all the signals are in the low state. When both inputs `in1` and `in2` go high, the output `out` also switches to logical 1. It stays in this state until both inputs go low, at which stage the output switches to logical 0.



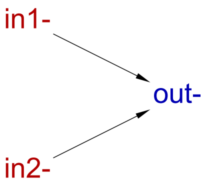


Modelling

Let us model the C-element behaviour using the *Signal Transition Graph (STG)* formalism. Create a new STG work called *stg-celement* and translate the sequence of events in the timing diagram into a sequence of STG transitions. Basically you need to create a signal transition for each event of the timing diagram and capture the causality between these events by means of directed arcs between signal transitions. Recreate the following STG model in Workcraft.

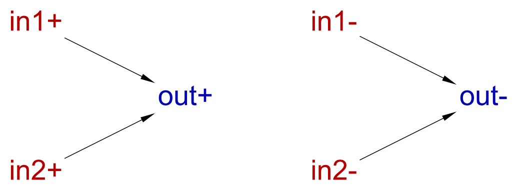
Start with the reset phase of the C-element where `out-` event is caused by `in1-` and `in2-` events:

- Activate the **signal transition generator** . Pay attention to the hint at the bottom of the screen: *Click to create a falling (or rising with* Ctrl *) transition of an output (or input with* Shift *) signal.*
- Click the Editor panel in the location where you want the `out-` transition to appear.
- Hold Shift and click in the desired location of `in1-` transition. The default name for the input signal is `in`, therefore you need to rename it. Activate the **selection tool** and click on the transition and in the Property editor change its *Name* from `in` to `in1`. Alternatively, double-click on the transition and change its name directly in the Editor.
- Create `in2-` transition the same way as `in1-`.
- Activate **connection tool** . Pay attention to the hints at the bottom of the screen. Initially the hint says *Click on the first component.* After the first component is chosen, the hint changes to *Click on the second component or create a node point. Hold* Ctrl *to connect continuously*.
- Click `in1-`, move the mouse pointer to the `out-` transition and click again. A causality arc from `in1-` to `out-` will be created.
- Repeat the connection procedure to capture causality between `in2-` and `out-` transitions.



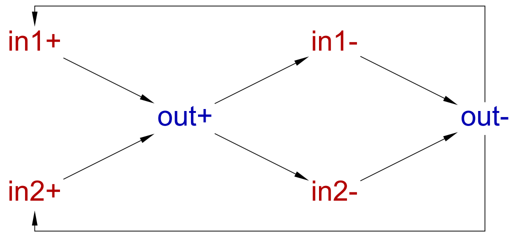
Similarly capture the set phase of the C-element:

- Create transitions `in1+`, `in2+` and `out+`. As before, rename the default `in` signal to `in1` and `in2` respectively.
- Create causality arcs from `in1+` to `out+` and from `in2+` to `out+`.
- Use the **selection tool** to rearrange the STG nodes similar to the following figure.



Now connect the set and reset portions of the specification:

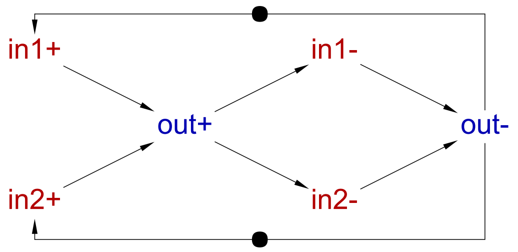
- The same as before, use **connection tool** to introduce causality from `out+` to `in1-` and to `in2-`.
- Create an arc from `out-` to `in1+`, but this time instead of a straight line connection use a polyline. After choosing the source of the connection (`out-` transition) click aside of the existing transitions - this will create a node point of a polyline. Continue forming the shape of the polyline by creating its node points, and finally choose the destination transition `in1+`.
- Repeat the same procedure for creating a polyline connection from `out-` to `in2+`.



Finally, specify the initial state of the C-element in accordance to the starting point of the timing diagram:

- Activate the **selection tool** .
- Select the arc from `out-` to `in1+` and in the Property editor set the *Tokens* value to 1. Alternatively, hold Ctrl and double-click this arc.
- Similarly put a token on the arc from `out-` to `in2+`.

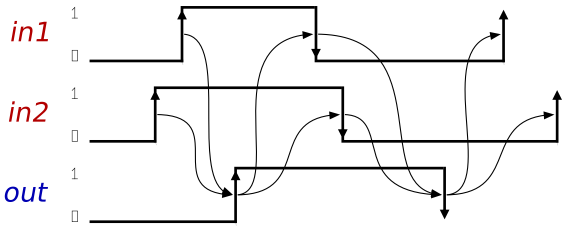
The resulting STG should look as follows.



Validation and verification of specification

Activate the **simulation tool** and exercise the obtained STG model. Click one of the enabled signal transitions (they are highlighted in orange) to *fire* (or *execute*) it, which changes the *state* (or *marking*) of the STG. Note that the sequence of fired transitions is recorded in the simulation trace that is somewhat similar to the original timing diagram. Check that the simulation traces correspond to the intended behaviour of C-element.

For example, simulate the following trace `in2+, in1+, out+, in1-, in2-, out-, in1+, in2+` – you can copy the trace to clipboard and paste it in simulation tool by pressing . Then use the generator of trace diagram to convert the STG simulation trace to *Digital Timing Diagram* model. The produced timing diagram should resemble the initial informal specification and looks as follows.



The visibility and color of signals in the generated timing diagram are defined in **Visible** and **Color** columns of the signal state table, at the bottom of **Tool controls** for STG simulation tool. The order of signal waveforms is also defined in this table by rearranging (drag-and-drop) the signal rows.

The timing diagram is built for the main simulation trace – **Trace** column. If the **Trace** is empty then the timing diagram generator will suggest to use **Branch** simulation results instead.

Before proceeding to the synthesis of the C-element it is a good idea to verify that its specification meets essential requirements, e.g. that it is consistent, free from deadlocks, input-proper and output-persistent. These properties can be checked via the *Verification* menu. Note that it is possible to check these properties either separately or jointly with a single click.

If the specification violates any of these properties then a trace leading to the problematic state will be reported. This trace can be simulated for better understanding the reported issues and for correcting them in the specification.

Synthesis

The STG specification can now be synthesised into an asynchronous circuit implementation either with MPSat or Petrify back-end tools via *Synthesis* menu.

A complex-gate implementation obtained with Petrify (*Synthesis→Complex gate [Petrify]* menu item) appears in the Output panel towards the end and is as follows: (Note that you may get a slightly different equation.)

```
[out] = in2 (in1 + out) + in1 out;
```

By opening the parenthesis one can re-write it as

```
[out] = in1 in2 + in2 out + in1 out;
```

This equation can be directly mapped into an AND-OR complex gate whose function is `Z = A*B + C*D +`

`E*F` ; let us call it *AO222* gate.

The association of the C-element ports to the *AO222* gate pins can be described by the following Verilog module:

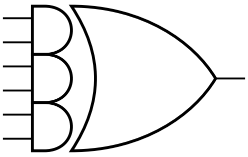
```
module celement (in1, in2, out);
  input in1;
  input in2;
  output out;
  AO222 inst_c (.A(in1), .B(in2), .C(in2), .D(out), .E(in1), .F(out), .Z(out));
endmodule
```

Circuit capturing

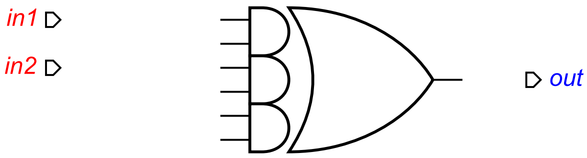
Workcraft can automatically derive a circuit from the synthesis output of MPSat or Petrify. (This is controlled by the *Import synthesis result* checkbox the *Edit→Preferences…* window under the *Models→Digital Circuit* leaf, and is enabled by default.) However, for the purposes of this exercise you should capture the circuit schematics manually.

Create a new Digital Circuit work called *circuit-celement-cg* and capture the implementation suggested by Petrify in form of a gate-level netlist.

- Activate **functional generator** and click in the desired position of the AND-OR complex gate.
- Activate **selection tool** .
- Select the only pin of the newly created function component.
- In the Property editor change the *Name* of the pin to `Z` and modify its *Set function* to `A*B+C*D+E*F`.

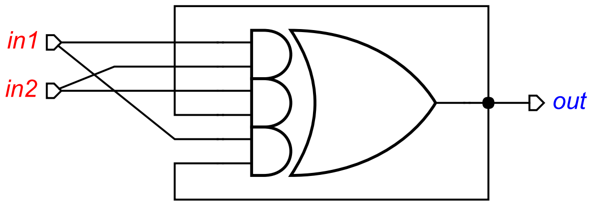


- Activate **port generator** . Pay attention to the hint at the bottom of the screen: *Click to create an output port (hold Shift for input port).*
- Click in intended location of the output port. Note that by default the port will be named `out1` - you can change this name to `out` later.
- Hold Shift and click in the desired locations of the input ports – they will be automatically assigned the names `in0` and `in1`.
- Switch to the selection tool. Choose the output port, go to the property editor and change port *Name* to `out`. Similarly change the name of input port `in0` to `in2`.

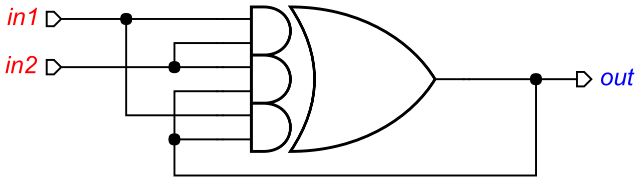


- Activate **connection tool** .

- Connect input ports `in1` to the 1st and 5th pins of the complex gate (`A` and `E` respectively).
- Connect input ports `in2` to the 2nd and 3rd pins of the complex gate (`B` and `C` respectively).
- Connect the output pin of the gate to the output port `out` and to the 4th and 6th inputs of the gate (`D` and `F` respectively).



You may wish to tidy up the circuit schematic and make it more readable by adding wire forks. If you create a connection to/from a wire then a **joint** will be automatically inserted. This result should look as follows; this implementation can be downloaded here: circuit-celement-cg.work (3.09 KiB).



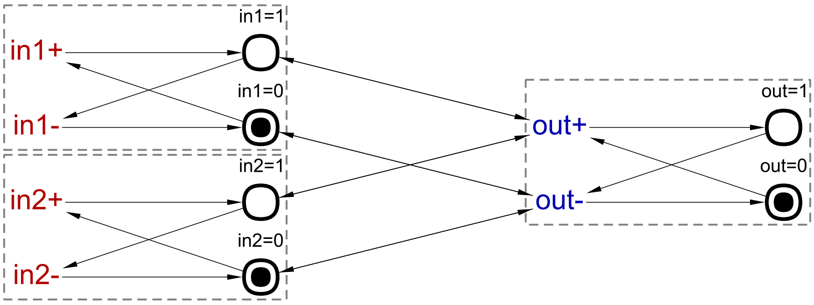
Verification of implementation

Activate the **simulation tool** ▶ and simulate the captured complex gate implementation of the C-element.

Ports, pins and wires are colour-coded: blue means low level and red means high level of the signal. Excited pins and ports are highlighted in orange.

Click one of the excited pins to toggle its logical value. Similar to the STG simulation, the sequence of signal events is recorded in the simulation trace and can be subsequently replayed for analysing the circuit's behaviour or converted to a timing diagram for visualisation purpose.

To conduct formal verification the circuit has to be converted into an STG. Normally this is done silently by the tool, but it is possible to view the intermediate circuit-STG via *Conversion→Signal Transition Graph*. The resulting STG for this circuit is shown below. Note that your layout may be slightly different and that this STG will not be required in the rest of the tutorial.



Note that if the C-element's inputs are not restricted in any way then they can change in an unexpected manner and cause malfunction of the circuit. This can be confirmed by checking the circuit for hazards *Verification→Hazards [MPSat]* – the circuit has a hazard after the following trace: `in2+, in1+`.

Play this trace to discover that indeed, at the end of it the output of the C-element is excited and ready to switch to logical 1, but an unexpected `in1-` or `in2-` transition would disable it, that does not fit the original STG specification.

The STG specification contains some information that is not available in the circuit, namely the behaviour of the environment. The circuit is guaranteed to work correctly only in an environment that respects its contract specified by the original STG. In other environments the circuit may exhibit hazards, deadlocks, etc. Therefore, during formal verification of the circuit one has to restrict its environment in such a way that the circuit only receives those inputs that are allowed by the STG specification.

This can be achieved as follows:

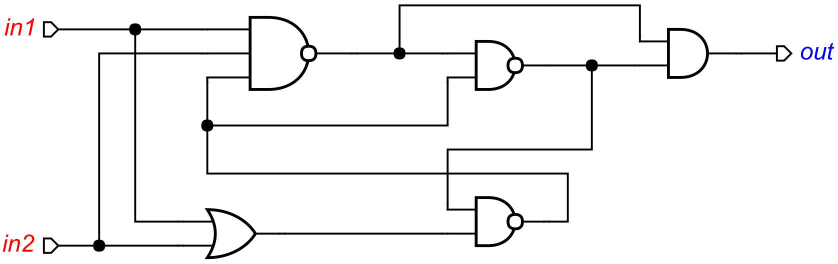
- In the circuit editor make sure that no components are selected (click on the editor canvas).
- In the property editor choose the *Environment URI* property and select the work file with the original STG specification.

Repeat the verification procedure to check that assuming a well behaved environment the circuit is free from hazards, deadlock-free and conforms to the environment specification. All these verification steps can be performed simultaneously via *Verification→Conformation, deadlock and hazard (reuse unfolding) [MPSat]*.

Decomposed implementation

Large gates like A0222 may not be available in the technology library and thus other implementations of C-element using smaller gates are of interest. However, it is very easy to make a mistake when designing asynchronous circuits, and so any such implementation has to be formally verified against the original STG specification.

Download the following C-element implementation due to Mayevsky circuit-celement-decomposed.work (3.78 KiB) and verify that it conforms to the STG specification and is free from deadlocks and hazards.



Note that the correctness of this implementation depends on the *isochronic forks assumption*: the difference in arrival times of a signal to the ends of a wire fork is negligible compared to any gate delay. If this assumption is violated, the above implementation can exhibit hazards. Download the following C-element implementation circuit-celement-decomposed-hazard.work (3.96 KiB) where a delay in a wire fork is made explicit (modelled by a buffer). Formally verify this model, inspect the violation trace, and explain what can go wrong in it.