6.4.3 Equivalence

Given the above reassurance that the implementation is apparently behaving in the right way, the equivalence proof between behaviours for the two versions of the cell may be completed. The proof in the HOL system devolves into a case analysis on time, and results in a theorem of the usual form:

$$
\begin{array}{l}
\vdash\ \forall clck\ outp\ ci\ co\ now.\\
\quad \lambda t.\ \left(\begin{array}{l}
\texttt{if}\ \big((t \geq now+1) \wedge clck \wedge ci\big)\ \texttt{then}\\
\quad \big\{(\mathrm{clck}, clck), (\mathrm{outp}, \neg outp), (\mathrm{ci}, ci), \big(\mathrm{co}, (ci \wedge \neg outp)\big)\big\}\\
\texttt{elseif}\ (t \geq now)\ \texttt{then}\\
\quad \big\{(\mathrm{clck}, clck), (\mathrm{outp}, outp), (\mathrm{ci}, ci), (\mathrm{co}, co)\big\}\\
else\ \{\ \}
\end{array}\right)\\
\qquad\qquad\qquad\qquad \overset{\tau}{=}\\
\qquad\qquad\qquad\qquad \underset{\{\mathrm{clck,outp,ci,o}\}}{}\\
\quad \lambda t.\ \texttt{if}\ \big((t \geq now+1) \wedge clck \wedge ci\big)\ \texttt{then}\\
\qquad \left\{\begin{array}{l}
(\mathrm{clck}, clck), \big(\mathrm{outp}, \big((ci \vee outp) \wedge (\neg ci \vee \neg outp)\big)\big),\\
(\mathrm{ci}, ci), \big(\mathrm{co}, (ci \wedge \neg outp)\big), (\mathrm{new}, outp)
\end{array}\right\}\\
\qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\\
\qquad \left\{\begin{array}{l}
(\mathrm{clck}, clck), (\mathrm{outp}, outp), (\mathrm{ci}, ci), (\mathrm{co}, co),\\
\big(\mathrm{new}, \big((ci \vee outp) \wedge (\neg ci \vee \neg outp)\big)\big)
\end{array}\right\}\\
\qquad \texttt{else}\ \{\ \}
\end{array}
$$

$$
\begin{array}{l}
\vdash \forall now\ clck\ outp\ ci\ co. \\
\quad \texttt{let}\ \Pi = \ldots\ \texttt{in}\ \mu \vdash \langle \Pi, \lambda t.\{\} \rangle \xrightarrow{sim} beh = \\
\quad beh = \lambda t.\ \texttt{if}\ \big((t \geq now + 1) \wedge clck \wedge ci\big)\ \texttt{then} \\
\qquad \left\{ \begin{array}{l} (\text{clck}, clck), \big(\text{outp}, \big((ci \vee outp) \wedge (\neg ci \vee \neg outp)\big)\big), \\ (\text{ci}, ci), \big(\text{co}, (ci \wedge \neg outp)\big), (\text{new}, outp) \end{array} \right\} \\
\qquad \texttt{elseif}\ (t \geq now)\ \texttt{then} \\
\qquad \left\{ \begin{array}{l} (\text{clck}, clck), (\text{outp}, outp), (\text{ci}, ci), (\text{co}, co), \\ \big(\text{new}, \big((ci \vee outp) \wedge (\neg ci \vee \neg outp)\big)\big) \end{array} \right\} \\
\qquad \texttt{else}\ \{\}
\end{array}
$$

Since the behaviour of the device contains longer Boolean expressions than those of earlier examples, it is not intuitively clear that the right result has been obtained. One can gain some confidence in the derived behaviour by specialising the variables which represent the values of the signals. The interesting behaviour only occurs when the clock is high. So by specialising the variable *clck* to high, the conditional is reduced to:

$$
\begin{array}{l}
\lambda t.\ \texttt{if}\ \big((t \geq now + 1) \wedge ci\big)\ \texttt{then} \\
\quad \left\{ \begin{array}{l} (\text{clck}, \mathsf{T}), \big(\text{outp}, \big((ci \vee outp) \wedge (\neg ci \vee \neg outp)\big)\big), \\ (\text{ci}, ci), \big(\text{co}, (ci \wedge \neg outp)\big), (\text{new}, outp) \end{array} \right\} \\
\quad \texttt{elseif}\ (t \geq now)\ \texttt{then} \\
\quad \left\{ \begin{array}{l} (\text{clck}, \mathsf{T}), (\text{outp}, outp), (\text{ci}, ci), (\text{co}, co), \\ \big(\text{new}, \big((ci \vee outp) \wedge (\neg ci \vee \neg outp)\big)\big) \end{array} \right\} \\
\quad \texttt{else}\ \{\}
\end{array}
$$

If one further assumes that the carry input is high, the behaviour simplifies to:

$$
\begin{array}{l}
\lambda t.\ \texttt{if}\ (t \geq now + 1)\ \texttt{then} \\
\quad \big\{(\text{clck}, \mathsf{T}), (\text{outp}, \neg outp), (\text{ci}, \mathsf{T}), (\text{co}, \neg outp), (\text{new}, outp)\big\} \\
\quad \texttt{elseif}\ (t \geq now)\ \texttt{then} \\
\quad \big\{(\text{clck}, \mathsf{T}), (\text{outp}, outp), (\text{ci}, \mathsf{T}), (\text{co}, co), (\text{new}, \neg outp)\big\} \\
\quad \texttt{else}\ \{\}
\end{array}
$$

One now observes that the result of the cell counts after 1 unit to the inverse of whatever the initial output was. Furthermore, new has had its own value inverted to remember the previous output of the cell. The carry output also becomes the inverse of the original output.

then conjoined with the carry input to produce the carry output. The output and carry input are also passed through an exclusive-or gate to create the next value for the counter. The next time the clock rises, this is the value that will be transmitted to the output. The delay through each of the combinational gates is zero.

The Femto-VHDL for the implementation of the cell may be read directly from the above schematic:

$$\begin{array}{l}\textsf{arch cell low}\ (\{\text{clck},\text{ci}\},\{\text{outp}\},\{\text{co}\})\ (\textsf{signal}\ \{\text{new}\}) \\ \quad \left(\begin{array}{l}\{\text{ci},\text{outp}\} : \left(\text{co} :\overset{\text{inert}}{=} (\text{ci and outp}, 0)\right) \ ||\ \{\text{ci},\text{outp}\} : \left(\text{new} :\overset{\text{inert}}{=} (\text{ci xor outp}, 0)\right) \ || \\ \{\text{clck}\} : \left((\text{clck}) \texttt{ => outp} :\overset{\text{inert}}{=} (\text{new}, 1) \mid \textsf{null}\right)\end{array}\right)\end{array}$$

The first two processes are simple variations of the combinational gates used in previous examples. The third process represents the flip-flop. Whenever the clock is active and high (a rising edge), the input is conveyed to the output after 1 unit. The state of the output remains unchanged in all other situations. The input to the flip-flop is the internal signal new. Its output is the current count result outp.

The term used to begin the symbolic simulation is almost identical to that used for the specification of the cell.

$$\begin{array}{l}\texttt{"let}\ \Pi = \textsf{dflat}\left(\textsf{arch cell low}\ (\{\text{clck},\text{ci}\},\{\text{outp}\},\{\text{co}\})\ (\textsf{signal}\ \{\text{new}\})\ \ldots\right) \\ \quad \texttt{and}\ \sigma = \left\{(\text{clck}, clck),(\text{outp}, outp),(\text{ci}, ci),(\text{co}, co),\left(\text{new},\left((ci \vee outp) \wedge (\neg ci \vee \neg outp)\right)\right)\right\}\ \texttt{in} \\ \quad \texttt{let}\ \mu = \left(\textsf{get_delays_cs}\ \Pi, now, \sigma, \{\text{clck}\}, \lambda t.(\{\},\{\})\right)\ \texttt{in} \\ \qquad \mu \vdash \left\langle \Pi, \lambda t.\{\}\right\rangle \xrightarrow{sim} beh\texttt{"}\end{array}$$

The only addition is the starting information for the internal signal new. The general behaviour of the parity checker implementation showed that the initial values of such signals need to based on those of externally visible ones. Otherwise, there is no common basis for comparison between two versions of the same device. Here new is given an initial value in terms of the exclusive-or of the output and carry input of the cell.

The behaviour of the implementation follows in a similar fashion to the previous examples:

The initial σ makes no special assumptions about the interrelationships of the various signals, and the starting event is on the clock.

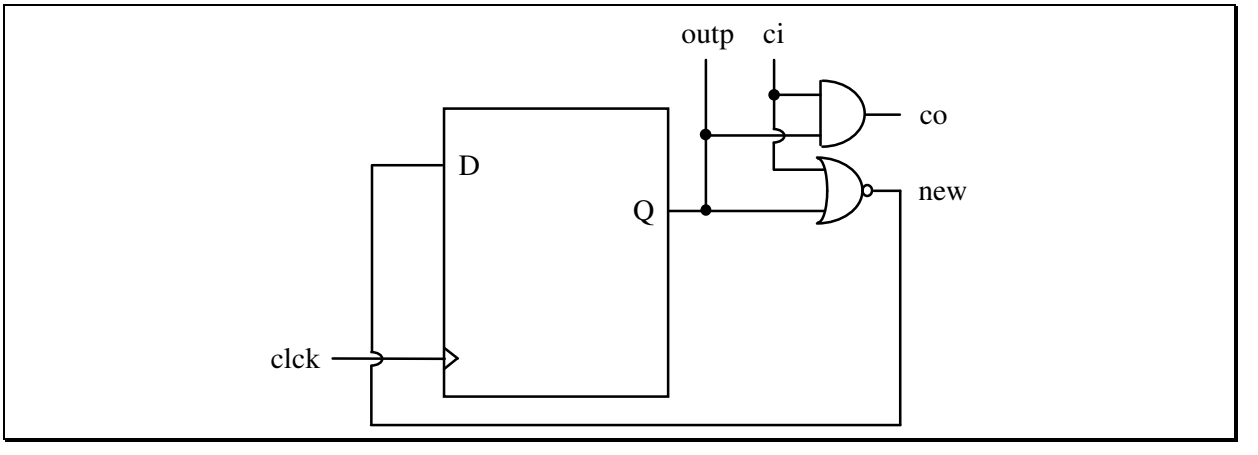
The HOL theorem that emerges after the symbolic simulation demonstrates the expected behaviour for the cell.

$$
\begin{array}{l}
\vdash \forall now\ clck\ outp\ ci\ co.\\
\quad \texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ cell\ high}\ \left(\{\mathrm{clck},\mathrm{ci}\},\{\mathrm{outp}\},\{\mathrm{co}\}\right)\left(\mathsf{signal}\ \{\}\right)\ldots\right)\\
\quad \texttt{and}\ \sigma = \left\{(\mathrm{clck},clck),(\mathrm{outp},outp),(\mathrm{ci},ci),(\mathrm{co},co)\right\}\ \texttt{in}\\
\quad \texttt{let}\ \mu = \ldots\ \texttt{in}\\
\qquad \mu \vdash \left\langle \Pi, \lambda t.\{\}\right\rangle \xrightarrow{sim} beh =\\
\left(beh = \lambda t.\left(\begin{array}{l}
\texttt{if}\ \left((t \geq now+1) \wedge clck \wedge ci\right)\ \texttt{then}\\
\quad \left\{(\mathrm{clck},clck),(\mathrm{outp},\neg outp),(\mathrm{ci},ci),(\mathrm{co},(ci \wedge \neg outp))\right\}\\
\texttt{elseif}\ (t \geq now)\ \texttt{then}\\
\quad \left\{(\mathrm{clck},clck),(\mathrm{outp},outp),(\mathrm{ci},ci),(\mathrm{co},co)\right\}\\
else\ \{\}
\end{array}\right)\right)
\end{array}
$$

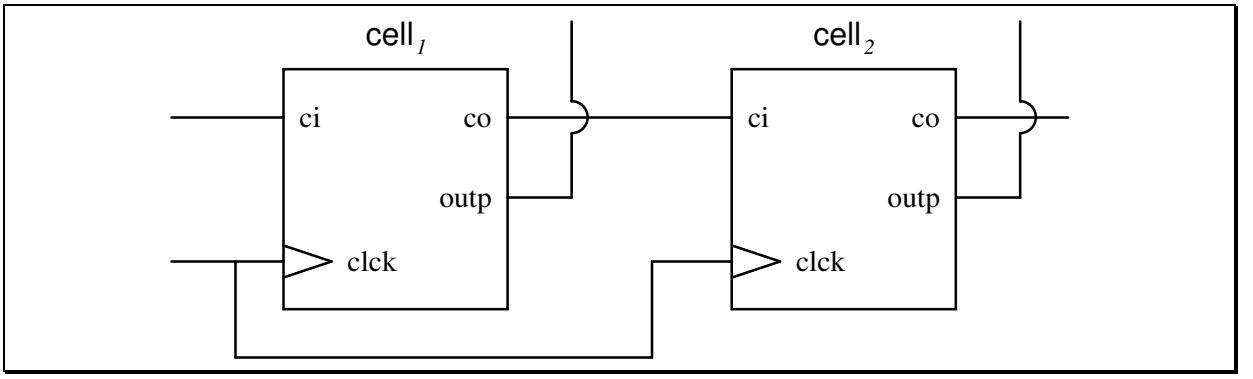
Whenever both the clock and the carry input are high, the output is inverted after one unit of time. Otherwise, nothing changes.

6.4.2 Implementation

The previous architecture presented an algorithmic view of the behaviour of the cell. The following diagram shows how it might be realised at the gate level.



The central component is a D-type flip-flop, and it is assumed to have a one unit transmission time from the input D to the output Q whenever the clock has a rising edge. The output of the cell is taken directly from the output of the flip-flop. It is



Each cell has four signals – a clock input clck, a carry input ci, a carry output co and an output outp showing the current state of the count. The clock synchronises the counter cells, and the overall device should be active on the rising edge of the clock. Furthermore, a cell should only count when its carry input is high. The carry output is intended to become high when an individual cell counts up to a high value.

6.4.1 Specification

The Femto-VHDL specification for one cell may easily be extracted from the above natural language description:

$$\text{arch cell high } (\{\text{clck},\text{ci}\},\{\text{outp}\},\{\text{co}\})\,(\text{signal}\,\{\,\})\left(\begin{array}{l}\{\text{ci},\text{outp}\}:\left(\text{co}:\overset{\text{inert}}{=}(\text{ci and outp},0)\right)\ ||\\ \{\text{clck}\}:\left((\text{clck and ci})\ \texttt{=>}\ \text{outp}:\overset{\text{inert}}{=}(\text{not outp},1)\ \texttt{|}\ \text{null}\right)\end{array}\right)$$

The input ports are made up of the clock clck and the carry input ci, the output of the device outp is a bi-directional port and the carry output co is a pure output port. The clocked process inverts the output after 1 unit when the carry input is high and a rising edge is detected. The carry output is calculated in the first process, and will only become high when the carry input and the output of the counter are both high.

The derivation of a general behaviour for this specification of the cell begins in an analogous manner to the previous examples. The term to be used is as expected:

$$\begin{array}{l}\text{"}\texttt{let}\ \Pi = \text{dflat}\left(\text{arch cell high}\ (\{\text{clck},\text{ci}\},\{\text{outp}\},\{\text{co}\})\,(\text{signal}\,\{\,\})\ldots\right)\\ \quad\texttt{and}\ \sigma = \{(\text{clck},\mathit{clck}),(\text{outp},\mathit{outp}),(\text{ci},\mathit{ci}),(\text{co},\mathit{co})\}\ \texttt{in}\\ \quad\texttt{let}\ \mu = \left(\text{get_delays_cs}\ \Pi,\mathit{now},\sigma,\{\text{clck}\},\lambda t.(\{\},\{\})\right)\ \texttt{in}\\ \qquad \mu \vdash \langle \Pi,\lambda t.\{\}\rangle \xrightarrow{\ sim\ } \mathit{beh}\text{"}\end{array}$$

If one abstracts away the internal signals and re-interprets time intervals, the following function on time is really what is being dealt with during the equivalence check:

$$
\begin{array}{l}
\lambda t.\,\texttt{if}\ inp\ \texttt{then} \\
\quad \texttt{if}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{inp}, inp), (\mathrm{outp}, \neg outp)\} \\
\quad \texttt{else}\ \{(\mathrm{inp}, inp), (\mathrm{outp}, outp)\} \\
\texttt{else}\ \{(\mathrm{inp}, inp), (\mathrm{outp}, outp)\}
\end{array}
$$

A trivial case analysis during the proof of the equivalence yields the expected theorem in the HOL system:

$$
\begin{array}{l}
\vdash\ \forall inp\ outp\ now. \\
\quad \lambda t.\,\texttt{if}\ (inp \wedge (t \geq now))\ \texttt{then}\ \{(\mathrm{inp}, inp), (\mathrm{outp}, \neg outp), (\mathrm{ini}, \mathsf{T})\} \\
\qquad \texttt{else}\ \{(\mathrm{inp}, inp), (\mathrm{outp}, outp), (\mathrm{ini}, \mathsf{T})\} \\
\qquad\qquad\qquad\qquad \overset{\tau}{=}_{\{\mathrm{inp,outp}\}} \\
\quad \lambda t.\,\texttt{if}\ inp\ \texttt{then} \\
\qquad \texttt{if}\ (t \geq now + 1)\ \texttt{then} \\
\qquad\quad \{(\mathrm{inp}, inp), (\mathrm{outp}, \neg outp), (11, outp), (13, \neg outp), (14, \mathsf{T}), (15, \mathsf{T})\} \\
\qquad \texttt{elseif}\ (t \geq now)\ \texttt{then} \\
\qquad\quad \{(\mathrm{inp}, inp), (\mathrm{outp}, \neg outp), (11, \neg outp), (13, \neg outp), (14, \mathsf{T}), (15, \mathsf{T})\} \\
\qquad \texttt{else}\ \{(\mathrm{inp}, inp), (\mathrm{outp}, outp), (11, \neg outp), (13, outp), (14, \mathsf{T}), (15, \mathsf{T})\} \\
\quad\ \texttt{else}\ \{(\mathrm{inp}, inp), (\mathrm{outp}, outp), (11, \neg outp), (13, outp), (14, \mathsf{T}), (15, \mathsf{T})\}
\end{array}
$$

6.4 A Counter Cell

The parity checker introduced a non-standard state-holding device to ensure quiescence of the design. The study presented here deals with the more traditional notion of state that is exemplified by one cell of a binary counter. The following diagram shows how two such cells may be put together in series make up a two-bit counter.

as well as that of the implementation:

$$\begin{array}{l}\lambda t.\,\texttt{if}\ (t \geq 1)\ \texttt{then}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\mathsf{T})\}\\ \quad \texttt{elseif}\ (t \geq 0)\ \texttt{then}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\bot)\}\\ \quad \texttt{else}\ \{\}\end{array}$$

at initialisation. The behaviour of the implementation may further be reduced to:

$$\begin{array}{l}\lambda t.\,\texttt{if}\ (t \geq 1)\ \texttt{then}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\mathsf{T})\}\\ \quad \texttt{else}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\bot)\}\end{array}$$

for the simple reason that all the possibilities have been covered by the first two branches of the conditional. The equivalence follows as a HOL theorem from the definition of $\overset{\tau}{=}$ on the signals inp and outp after a case analysis of the behaviour of the implementation.

$$\begin{array}{l}\vdash\ \lambda t.\,\{(\text{inp},\bot),(\text{outp},\mathsf{T})\}\\ \qquad\qquad \underset{\{\text{inp,outp}\}}{\overset{\tau}{=}}\\ \quad \lambda t.\,\texttt{if}\ (t \geq 1)\ \texttt{then}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\mathsf{T})\}\\ \qquad \texttt{else}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\bot)\}\end{array}$$

The second check that needs to be made is for subsequent iterations of the simulation loop. The behaviour of the specification in this instance is:

$$\begin{array}{l}\lambda t.\,\texttt{if}\ (\mathit{inp} \wedge (t \geq \mathit{now}))\ \texttt{then}\ \{(\text{inp},\mathit{inp}),(\text{outp},\neg\mathit{outp}),(\text{ini},\mathsf{T})\}\\ \quad \texttt{else}\ \{(\text{inp},\mathit{inp}),(\text{outp},\mathit{outp}),(\text{ini},\mathsf{T})\}\end{array}$$

while that derived for the implementation was:

$$\begin{array}{l}\lambda t.\,\texttt{if}\ \mathit{inp}\ \texttt{then}\\ \qquad \texttt{if}\ (t \geq \mathit{now}+1)\ \texttt{then}\\ \qquad\quad \{(\text{inp},\mathit{inp}),(\text{outp},\neg\mathit{outp}),(11,\mathit{outp}),(13,\neg\mathit{outp}),(14,\mathsf{T}),(15,\mathsf{T})\}\\ \qquad \texttt{elseif}\ (t \geq \mathit{now})\ \texttt{then}\\ \qquad\quad \{(\text{inp},\mathit{inp}),(\text{outp},\neg\mathit{outp}),(11,\neg\mathit{outp}),(13,\neg\mathit{outp}),(14,\mathsf{T}),(15,\mathsf{T})\}\\ \qquad \texttt{else}\ \{(\text{inp},\mathit{inp}),(\text{outp},\mathit{outp}),(11,\neg\mathit{outp}),(13,\mathit{outp}),(14,\mathsf{T}),(15,\mathsf{T})\}\\ \quad \texttt{else}\ \{(\text{inp},\mathit{inp}),(\text{outp},\mathit{outp}),(11,\neg\mathit{outp}),(13,\mathit{outp}),(14,\mathsf{T}),(15,\mathsf{T})\}\end{array}$$

"let Π = dflat (arch impl parity ({inp},{outp},{}) (signal {}) ...)
and $\sigma = \{(\text{inp}, inp), (\text{outp}, outp), (11, \neg outp), (13, outp), (14, \mathsf{T}), (15, \mathsf{T})\}$ in
let $\mu = (\text{get_delays_cs}\ \Pi, now, \sigma, \{\text{inp}\}, \lambda t.(\{\},\{\}))$ in
$\mu \vdash \langle \Pi, \lambda t.\{\} \rangle \xrightarrow{sim} beh$"

Why were these particular initial values for the signals chosen? Clearly there need to be two independent values of interest – the input signal value and the output signal value. If this course is chosen, then 11 will be the inverse of whatever the output is. Since 13 is being directly piped into the output, they both have the same value. Furthermore, it is assumed that time *now* occurs after initialisation has been completed. Both 14 and 15 therefore hold high values. One also notices that if the concrete value of outp was high and that of inp was low, the starting state of the signals corresponds to the initialisation behaviour at times greater than or equal to 1. The trace of past activity has been left empty as there are no instances of the 'delayed attribute that need to be catered for.

The derivation of the behaviour proceeds in a similar manner to earlier examples. The result is a bit more complex. The behaviour is however easily discernible from the nested conditionals.

$\vdash \forall now\ inp\ outp\ beh.$
  let ... in $\mu \vdash \langle \Pi, \lambda t.\{\} \rangle \xrightarrow{sim} beh =$
  $beh =$
    $\lambda t.$ if inp then
      if $(t \geq now + 1)$ then
        $\{(\text{inp}, inp), (\text{outp}, \neg outp), (11, outp), (13, \neg outp), (14, \mathsf{T}), (15, \mathsf{T})\}$
      elseif $(t \geq now)$ then
        $\{(\text{inp}, inp), (\text{outp}, \neg outp), (11, \neg outp), (13, \neg outp), (14, \mathsf{T}), (15, \mathsf{T})\}$
      else $\{(\text{inp}, inp), (\text{outp}, outp), (11, \neg outp), (13, outp), (14, \mathsf{T}), (15, \mathsf{T})\}$
    else $\{(\text{inp}, inp), (\text{outp}, outp), (11, \neg outp), (13, outp), (14, \mathsf{T}), (15, \mathsf{T})\}$

6.3.3 Equivalence

Two different equivalence checks will have to be made – one for initialisation, the other for subsequent iterations of the simulation loop. If one takes the initialisation phase first, it is necessary to recall both the behaviour of the specification:

$\lambda t. \{(\text{inp}, \bot), (\text{outp}, \mathsf{T}), (\text{ini}, \mathsf{T})\}$

Initialisation

The initialisation phase of the analysis of the implementation may now proceed. The procedure is much the same as with the specification of the device. Recall that the term representing the architecture was passed to the function simulate. The same is true here, with the exception that a singleton set containing the initial value of the signal l4 (high) is supplied as the second argument.

Once the initialisation conversion has been executed, the resulting projected behaviour (τ) is as follows. The calls to inert_post seen in the analysis of the specification have been expanded and simplified for readability:

$$\begin{array}{l}\lambda t.\,\texttt{if}\ (t=0)\ \texttt{then}\ \{(\text{outp},\mathsf{T}),(13,\bot)\}\\ \quad\texttt{elseif}\ (t=1)\ \texttt{then}\ \{(11,\mathsf{T}),(15,\mathsf{T})\}\\ \quad\texttt{else}\ \{\}\end{array}$$

The rest of the simulation may now be completed in an analogous manner to that of the specification. The resulting theorem in the HOL system appears as:

$$\begin{array}{l}\vdash \text{simulate}\ \Pi\ \{(14,\mathsf{T})\} =\\ \quad\forall beh.\ beh = \lambda t.\,\texttt{if}\ (t\geq 1)\ \texttt{then}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\mathsf{T})\}\\ \qquad\qquad\texttt{elseif}\ (t\geq 0)\ \texttt{then}\ \{(\text{inp},\bot),(\text{outp},\mathsf{T}),(11,\bot),(13,\mathsf{T}),(14,\mathsf{T}),(15,\bot)\}\\ \qquad\qquad\texttt{else}\ \{\}\end{array}$$

An analysis of this result shows that it meets the behavioural constraints for time 0 set down in the natural language specification. The output at time 0 is high – a value which is maintained throughout. Furthermore, no reference is made to any intervening δ-steps. One should also note that the value of the selector for the second multiplexor (15) changes from low to high after 1 unit of time has passed.

General Behaviour

Having derived the initialisation behaviour, one must now come up with an overall understanding of behaviour for the implementation. This presages a return to an environment that is replete with variables rather than concrete values. The starting term in the HOL system appears as:

form of mux. The version that will be used in the design is actually a form of clocked register:

$$\text{arch clkd reg}\left(\{sel, in_1, in_2\},\{\},\{outp\}\right)\left(\text{signal}\{\}\right)\left(\{sel\}:\left(\begin{array}{r}(sel) \texttt{=>}\ outp \overset{\text{inert}}{:=} (in_1, 0)\\ \texttt{|}\ outp \overset{\text{inert}}{:=} (in_2, 0)\end{array}\right)\right)$$

The component ensures that in_1 is only chosen when a rising edge is detected on the selector. This is not a true multiplexor, and could not be used in place of the previous version as the second mux in the system.

The whole design may be put together in the following fashion. The ellipsis again indicates where the concurrent statements associated with each architecture should go.

$$\begin{array}{l}\text{arch impl parity}\left(\{\text{inp}\},\{\},\{\text{outp}\}\right)\left(\text{signal}\{\}\right)\\ \quad\text{dflat}\left(\text{dpar}\left(\begin{array}{l}\text{dpar}\left(\begin{array}{l}\text{arch simple regstr}\left(\{14\},\{\},\{15\}\right)\left(\text{signal}\{\}\right)\ldots\\ \text{arch simple inv}\left(\{\text{outp}\},\{\},\{11\}\right)\left(\text{signal}\{\}\right)\ldots\end{array}\right)\\ \text{dpar}\left(\begin{array}{l}\text{arch simple mux}\left(\{\text{inp},11,\text{outp}\},\{\},\{13\}\right)\left(\text{signal}\{\}\right)\ldots\\ \text{arch clkd reg}\left(\{15,13,14\},\{\},\{\text{outp}\}\right)\left(\text{signal}\{\}\right)\ldots\end{array}\right)\end{array}\right)\right)\end{array}$$

If one is not concerned about the oscillation problem. The instantiation of the components would appear as:

$$\begin{array}{l}\text{arch impl parity}\left(\{\text{inp}\},\{\},\{\text{outp}\}\right)\left(\text{signal}\{\}\right)\\ \quad\text{dflat}\left(\text{dpar}\left(\begin{array}{l}\text{dpar}\left(\begin{array}{l}\text{arch simple regstr}\left(\{14\},\{\},\{15\}\right)\left(\text{signal}\{\}\right)\ldots\\ \text{arch simple inv}\left(\{\text{outp}\},\{\},\{11\}\right)\left(\text{signal}\{\}\right)\ldots\end{array}\right)\\ \text{dpar}\left(\begin{array}{l}\text{arch simple mux}\left(\{\text{inp},11,\text{outp}\},\{\},\{13\}\right)\left(\text{signal}\{\}\right)\ldots\\ \text{arch simple mux}\left(\{15,13,14\},\{\},\{\text{outp}\}\right)\left(\text{signal}\{\}\right)\ldots\end{array}\right)\end{array}\right)\right)\end{array}$$

Again, there are two distinct phases of simulation that must be undertaken with the implementation-level version of the parity checker – initial and general behaviour. The complimentary correspondence between these two forms of behaviour that was illustrated when working with the specification must also be shown here.

through, and the work is done by the first mux. When the input inp is high, the inverted value of the output (delayed by 1 unit) is selected. Otherwise, the output remains unchanged.

In the translation of these components into VHDL, it must be remembered that the "D" in VHDL stands for "description". One is modelling a physical device, not providing an *actualité*, and should therefore beware of situations where simulation may give rise to unexpected behaviour in an apparently obvious situation. The problem with the parity checker is that unless the first multiplexor is equipped with an edge-triggered selector, it is possible to leave the input high and create a unit-delay oscillator. As a result, one must either make sure that the input to the parity checker finishes with a low value, or build into it the edge-triggered behaviour just alluded to. This is because an oscillator, of course, never quieseces, and the basis for reasoning here is constrained to only those devices which quiesce. The realisation of the above schematic must therefore take this potential for oscillation into account.

The components used in the parity checker are not at all complicated. The simplest of them is the buffer:

$$\text{arch simple regstr } (\{inp\},\{\},\{outp\})\,(\text{signal }\{\})\left(\{inp\} : outp :=^{\text{inert}} (inp,1)\right)$$

Note that the signal names are variables. This is because the component has not been given an instantiation in a particular design. The functionality is as expected – the output is assigned the value of the input delayed by one unit. The inverter does much the same thing, with the exception that the delayed signal is inverted before the assignment takes place:

$$\text{arch simple inv } (\{inp\},\{\},\{outp\})\,(\text{signal }\{\})\left(\{inp\} : outp :=^{\text{inert}} (\text{not } inp,1)\right)$$

A multiplexor might reasonably be expected to take on the following form:

$$\text{arch simple mux } (\{sel,in_1,in_2\},\{\},\{outp\})\,(\text{signal }\{\})$$
$$\left(\{sel,in_1,in_2\} : \left((sel) \Rightarrow outp :=^{\text{inert}} (in_1,0) \mid outp :=^{\text{inert}} (in_2,0)\right)\right)$$

This is perfectly suitable for the second mux in the schematic (i.e. the one that eventually becomes a pass-through device). It is, however, unacceptable if one is not particularly concerned with the values being used as input to the parity checker. As mentioned above, it is entirely possible to create an oscillator with this particular

starting state of the signals would correspond to the just-derived initialisation behaviour.

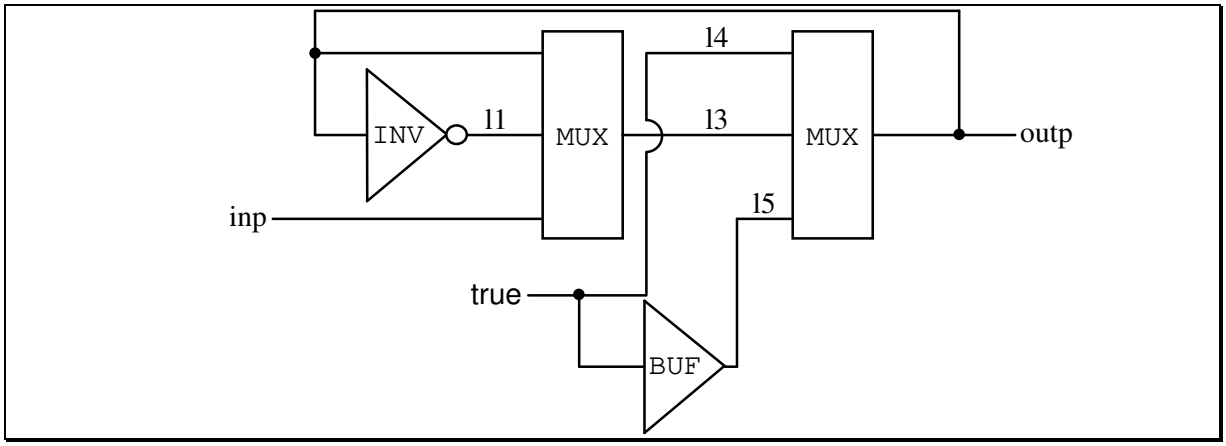
The computation of the general behaviour proceeds in a like manner to the earlier examples. At the end of the process, the following HOL theorem emerges:

$$
\begin{array}{l}
\vdash \forall now\ inp\ outp\ beh. \\
\quad \texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ parity\ high}\ (\{\mathrm{inp}\},\{\},\{\mathrm{outp}\})\ (\mathsf{signal}\ \{\mathrm{ini}\})\ \ldots\right) \\
\quad \texttt{and}\ \sigma = \{(\mathrm{inp}, inp),(\mathrm{outp}, outp),(\mathrm{ini},\mathsf{T})\}\ \texttt{in} \\
\quad \texttt{let}\ \mu = \ldots\ \texttt{in} \\
\qquad \mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh = \\
\left(beh = \lambda t. \begin{pmatrix} \texttt{if}\ (inp \wedge (t \geq now))\ \texttt{then}\ \{(\mathrm{inp}, inp),(\mathrm{outp}, \neg outp),(\mathrm{ini},\mathsf{T})\} \\ \texttt{else}\ \{(\mathrm{inp}, inp),(\mathrm{outp}, outp),(\mathrm{ini},\mathsf{T})\} \end{pmatrix} \right)
\end{array}
$$

The result is again as expected. The output immediately inverts when the input has a high value. Otherwise, the signal values do not change.

6.3.2 Implementation

The implementation of the device is made up of four components. It includes delay elements, as well as an external feedback loop. The original design was given in [18], and has been slightly simplified here for clarity. The schematic for the device is:



The delay through each multiplexor component is zero, while that through both the buffer and the inverter is 1 unit. The signal l4 is initialised to a high value, and all other signals start low. This ensures that the device will output a high value between power-up and the first unit of simulation time, thus meeting the original specification through the second mux. After time 1, this mux becomes a pass-

The initialisation part is animated by RUNINIT_CONV, whose behaviour is similar to that of RUNCYCLE_CONV described in Chapter 5. It merely executes the rules of the initialisation sequence from Chapter 4, Section 4.3.4. The result of initialisation is the derived transaction:

$$\text{inert_post ini } \bot\ (\lambda t.\ \{\})\ 0\ 0$$

When used as input to the simulation loop, this transaction filters through the program in a δ-step to give the following HOL theorem (Π denotes the program):

$$\vdash \text{simulate } \Pi\ \{\} = \forall beh.\ beh = \lambda t.\ \texttt{if}\ (t \geq 0)\ \texttt{then}\ \{(\text{outp}, \mathsf{T}), (\text{inp}, \bot), (\text{ini}, \mathsf{T})\}\ \texttt{else}\ \{\}$$

which is really:

$$\vdash \text{simulate } \Pi\ \{\} = \forall beh.\ beh = \lambda t.\ \{(\text{outp}, \mathsf{T}), (\text{inp}, \bot), (\text{ini}, \mathsf{T})\}$$

and shows that the program behaves in the expected way at time 0.

General Behaviour

The process of deriving a general behaviour for the specification is begun in an analogous manner to the studies in Sections 6.1 and 6.2. The term used in the HOL system to start the symbolic simulation is:

```
"let Π = dflat (arch parity high ({inp},{},{outp}) (signal {ini}) ...)
 and σ = {(inp, inp), (outp, outp), (ini, T)} in
 let μ = (get_delays_cs Π, now, σ, {inp}, λt. if (t ≤ now − 1) then ({}, σ) else {}) in
     μ ⊢ ⟨Π, λt. {}⟩ —sim→ beh"
```

The ellipsis shows where the concurrent statements in the program should appear. The simulation environment has been augmented by a trace θ giving the past values of the signals. It ensures that the 'delayed signal attribute used in the program works properly. The events in this θ have been left empty, but a set of signal-value pairs has been given and tied to the initial values of the signals. The signal ini has been given a high value as it is assumed that the simulation behaviour of interest occurs after the completion of the initialisation phase. The environment μ is also related to the initialisation behaviour. If *now* was 1, *inp* was low and *outp* was high, the

The above architecture represents an idealised version of what is desired. In Femto-VHDL, the user is not able to access the global clock (which is possible in full VHDL). The version that will be used in the case study makes use of the initialisation sequence of a simulation to accomplish the same task. The monolithic process above must now become two parallel processes.

```
arch parity high ({inp},{},{outp}) (signal {ini})
    {ini} : ( ini :=inert (true, 0) )  ||
    {inp, ini} : ( (ini'event) => outp :=inert (true, 0) |
                   (inp and not (ini'event)) => outp :=inert (not outp'delayed(1), 0) |
                   null )
```

Here the signal ini is used as a trigger to assign outp the value true at time 0. Recall the discussion of the initialisation process in Chapter 4 at the end of Section 4.3.5. Initialisation involves the waking up of all the processes in the program and running them once before handing control over to the main simulation loop. During initialisation of the current program, there are by definition no events on any of the signals, and both inp and ini have a default low value. Therefore, the only way that ini can have an event is as a result of the initialisation sequence waking up the first process and causing a high value to be immediately scheduled for it. The conditional in the second process will then assign a high value to outp one δ-step later once the simulation loop has been entered. The second process does nothing when initially activated due to the values of the signals all being low and the absence of any events. After the initialisation sequence is over ini is no longer a factor, and only the second process can ever execute.

Initialisation

In order to obtain the initialisation behaviour, the above program is given to the function simulate with an empty set of initial values. In doing so, it becomes necessary to perform two kinds of simulation. The first is different from previous examples in that it involves the initialisation sequence. The second is more familiar, and is about the execution of the simulation loop. The difference is that previous simulations began with empty future transactions. Here the results of initialisation will be used as the starting point.

6.3 Parity Checker

The above examples were sufficient to introduce basic concepts and techniques. A larger example is however required to demonstrate how concrete initialisation and quiescence restrictions are dealt with, as well as some subtleties of the semantics. A standard (albeit somewhat overworked) example from the HOL literature is a parity checker. The device was introduced in [18]. The basic premise is to have a device with one input and one output which satisfies the following two constraints:

- The output at time 0 is high.
- The output is high when an even number of highs have been received on the input.

The first requirement gives rise to a change from the kind of analysis that was performed in the previous examples. With the introduction of a specific kind of behaviour at time 0, one is forced to examine not only the general behaviour of the device, but also what it does when initialised.

6.3.1 Specification

The derivation begins, as in the earlier examples, with a program specifying the behaviour of the device in terms of a high-level VHDL design. The obvious Femto-VHDL program would be:

$$\text{arch parity high } (\{\text{inp}\},\{\},\{\text{outp}\})\ (\text{signal } \{\})$$

$$\{\text{inp}\} : \left(\begin{array}{l} (now = 0) \texttt{=>} \text{ outp} \overset{\text{inert}}{:=} (\text{true}, 0) \ \texttt{|} \\ \text{inp} \texttt{=>} \text{ outp} \overset{\text{inert}}{:=} (\text{not outp'delayed}(1), 0) \ \texttt{|} \\ \text{null} \end{array} \right)$$

In another change from previous examples, the ports include one of VHDL direction `inout`. This is because the signal in question, outp, is both read from and written to as part of a feedback loop. The conditional statement may be read directly from the informal specification (i.e. make sure the output is high at time zero as well as when an even number of high inputs have been counted). The inclusion of 1 as the argument to 'delayed assumes that the input signal being sampled has a period of 1 unit.

$$\vdash \forall now\ a\ b\ c\ beh.$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a \vee b))\} \\ \qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\} \\ \qquad \texttt{else}\ \{\} \end{array}\right)$$
$$\overset{\tau}{=}_{\{\mathrm{A},\mathrm{B},\mathrm{C}\}}$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg a \wedge \neg b)\} \\ \qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\} \\ \qquad \texttt{else}\ \{\} \end{array}\right)$$

6.2.3 Non-equivalence

It is also possible to show that the behaviours derived in Section 6.2.1 are *not* equivalent to those just given in Section 6.2.2. In each case the place where differences arise is in the value of signal C at times greater than or equal to $now+1$. As an example, consider the comparison of the negation of a conjunction and the conjunction of negations. If the derived behaviours for each are examined, the following theorem about their non-equivalence may be proved ($\overset{\tau}{\neq}$ should be understood to mean "not equivalent according to the relation $\overset{\tau}{=}$").

$$\vdash \forall now\ a\ b\ c\ beh.$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a \wedge b))\} \\ \qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\} \\ \qquad \texttt{else}\ \{\} \end{array}\right)$$
$$\overset{\tau}{\neq}_{\{\mathrm{A},\mathrm{B},\mathrm{C}\}}$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg a \wedge \neg b)\} \\ \qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\} \\ \qquad \texttt{else}\ \{\} \end{array}\right)$$

Similar results may be obtained for any permutation of comparisons of the behaviours in Sections 6.2.1 and 6.2.2. This correspondence to the expected behaviour of the DeMorgan property further increases one's confidence in the simulation semantics that Femto-VHDL has been given and the way that it is implemented in the HOL system.

$$\text{"}\texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ dm\ no}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\ldots\left(\{\mathrm{A},\mathrm{B}\} : \mathrm{C} \overset{\mathsf{inert}}{:=} \left(\mathsf{not}\ (\mathrm{A}\ \mathsf{or}\ \mathrm{B}),1\right)\right)\right)\ \texttt{in}$$
$$\texttt{let}\ \mu = \left(\mathsf{get_delays_cs}\ \Pi, now, \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\},\{\mathrm{A}\},\lambda t.(\{\},\{\})\right)\ \texttt{in}$$
$$\mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh\text{"}$$

$$\text{"}\texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ dm\ an}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\ldots\left(\{\mathrm{A},\mathrm{B}\} : \mathrm{C} \overset{\mathsf{inert}}{:=} \left(\mathsf{not\ A\ and\ not\ B},1\right)\right)\right)\ \texttt{in}$$
$$\texttt{let}\ \mu = \left(\mathsf{get_delays_cs}\ \Pi, now, \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\},\{\mathrm{A}\},\lambda t.(\{\},\{\})\right)\ \texttt{in}$$
$$\mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh\text{"}$$

Behaviour

Symbolic simulation of these terms leads to the following two theorems in the HOL system. The results are much the same as in Section 6.2.1.

$$\vdash \forall now\ a\ b\ c\ beh.$$
$$\texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ dm\ no}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\ldots\right)\ \texttt{in}\ \ldots =$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a \vee b))\} \\ \qquad\qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\} \\ \qquad\qquad \texttt{else}\ \{\} \end{array}\right)$$

$$\vdash \forall now\ a\ b\ c\ beh.$$
$$\texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ dm\ an}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\ldots\right)\ \texttt{in}\ \ldots =$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg a \wedge \neg b)\} \\ \qquad\qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\} \\ \qquad\qquad \texttt{else}\ \{\} \end{array}\right)$$

The difference between these simulations and the earlier ones is in the value of signal C for all times greater than or equal to $now+1$.

Equivalence

As in the previous version, the simulations may be shown to be $\overset{\tau}{=}$-equivalent:

$$\vdash \forall now\ a\ b\ c\ beh.$$
$$\texttt{let}\ \Pi = \text{dflat}\left(\text{arch dm on}\ (\{\text{A},\text{B}\},\{\text{C}\},\{\})\ \ldots\ \right)\ \texttt{in}\ \ldots =$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\text{A},a),(\text{B},b),(\text{C},\neg a \vee \neg b)\} \\ \qquad\quad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\text{A},a),(\text{B},b),(\text{C},c)\} \\ \qquad\quad \texttt{else}\ \{\} \end{array}\right)$$

Equivalence

After simulation, the equivalence of the behaviours for the two programs may be ascertained. The only signals of interest are again the input and output ports A, B and C (these also happen to be the only signals in either program). The proof is trivial, and involves comparing the various branches of the conditional statement in each behaviour. The only place where there is not an immediately discernible equality is in the comparison of $\neg(a \wedge b)$ and $(\neg a \vee \neg b)$ at all times greater than or equal to $now+1$. This turns out not surprisingly to be an instance of the DeMorgan property. The actual theorem proved in the HOL system is the following:

$$\vdash \forall now\ a\ b\ c\ beh.$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\text{A},a),(\text{B},b),(\text{C},\neg(a \wedge b))\} \\ \qquad\quad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\text{A},a),(\text{B},b),(\text{C},c)\} \\ \qquad\quad \texttt{else}\ \{\} \end{array}\right)$$
$$\overset{\tau}{=}_{\{\text{A,B,C}\}}$$
$$\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\text{A},a),(\text{B},b),(\text{C},\neg a \vee \neg b)\} \\ \qquad\quad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\text{A},a),(\text{B},b),(\text{C},c)\} \\ \qquad\quad \texttt{else}\ \{\} \end{array}\right)$$

In fact, since the signals are the same in both programs, and no internal declarations are being used, one could replace the use of $\overset{\tau}{=}$ with a simple equality to make a stronger statement.

6.2.2 Negation of a Disjunction vs. a Conjunction of Negations

The second half of the property, namely that the negation of a disjunction is the same as a conjunction of negations follows in an identical manner. The process again begins with the two HOL terms setting up the simulations:

take place. Each of them may be seen as the specification for the other, and vice versa. To that end, no hierarchical distinction will be made between the two programs being examined in each part of the study.

6.2.1 Negation of a Conjunction vs. a Disjunction of Negations

In an identical manner to the previous study, the analysis begins with terms in the HOL system which specify the simulation of the desired programs. For purposes of the current analysis, the following two are used:

$$\text{"let } \Pi = \text{dflat}\left(\text{arch dm na}\left(\{A,B\},\{C\},\{\}\right)\ldots\left(\{A,B\} : C \overset{\text{inert}}{:=} (\text{not (A and B)},1)\right)\right) \text{ in}$$
$$\text{let } \mu = \left(\text{get_delays_cs } \Pi, now, \{(A,a),(B,b),(C,c)\},\{A\},\lambda t.(\{\},\{\})\right) \text{ in}$$
$$\mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh\text{"}$$

$$\text{"let } \Pi = \text{dflat}\left(\text{arch dm on}\left(\{A,B\},\{C\},\{\}\right)\ldots\left(\{A,B\} : C \overset{\text{inert}}{:=} (\text{not A or not B},1)\right)\right) \text{ in}$$
$$\text{let } \mu = \left(\text{get_delays_cs } \Pi, now, \{(A,a),(B,b),(C,c)\},\{A\},\lambda t.(\{\},\{\})\right) \text{ in}$$
$$\mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh\text{"}$$

The first term represents the simulation of the negation of a conjunction, while the second deals with a disjunction of negations. As there are no local signals in the program, their declarations have been omitted for conciseness. Note that as in the previous study both environments are the same, and that the signals in each program begin the simulation with arbitrary values.

Behaviour

Each of these terms may be simulated in an identical manner to the specification of the NAND gate. Afterwards, the following theorems giving the behaviour of each program are arise:

finish_GAMMA, and is a reason why this particular kind of verification can never be a completely automatic method.

6.1.3 Equivalence

Having arrived at a behaviour for both versions of the NAND device, it is now possible to determine their equivalence. Recall that the way in which all the equivalence relations were defined in Chapter 4 imposed an obligation on the user to nominate signals of interest. In the present circumstances, the interest is in the behaviour of the device on the ports - in effect abstracting away internally visible signals. Given the definition of equivalence on transactions $\overset{\tau}{=}$, it is easy to prove the following theorem:

$$
\begin{array}{l}
\vdash\ \forall now\ a\ b\ c\ beh.\\
\quad \left(\begin{array}{l} beh = \lambda t.\,\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a\wedge b))\}\\ \qquad\quad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\}\\ \qquad\quad \texttt{else}\ \{\}\end{array}\right)\\
\qquad\qquad\qquad\qquad \overset{\tau}{=}_{\{\mathrm{A},\mathrm{B},\mathrm{C}\}}\\
\quad \left(\begin{array}{l} beh = \lambda t.\,\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a\wedge b)),(\mathrm{TMP},(a\wedge b))\}\\ \qquad\quad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c),(\mathrm{TMP},\neg(a\wedge b))\}\\ \qquad\quad \texttt{else}\ \{\}\end{array}\right)
\end{array}
$$

Essentially, all the definition of $\overset{\tau}{=}$ does is factor out any reference to the signal TMP, leaving two identical functions. The proof in the HOL system is actually performed via case analysis on time.

6.2 DeMorgan Property

The next example demonstrates a VHDL version of the DeMorgan property (i.e. that the negation of the conjunction of two Boolean terms is the same as the disjunction of the negation of each of the terms, and vice versa). The motivation behind the example is to not only show that it is possible to prove that different architectures implement various parts of the property, but also that some of their behaviours provably differ.

The distinction between specification and implementation is slightly blurred in this study. In each part of the analysis, a comparison of two uniprocess designs will

$now+1$. After the process performing the `and` has executed, the state of the signals is $\{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c),(\mathrm{TMP},a\wedge b)\}$. The same ambiguity about events seen in the specification also occurs here. This time, the question is about the value of TMP. The split introduced by `finish_GAMMA` presents the user with two execution paths. The first gives an event on TMP. The second does not. Clearly, an event is needed so that the second process may execute to assign a result to the output. Specialising the variable *tmp* to $\neg(a\wedge b)$ ensures that only the event branch is taken.

More iterations of `ONCE_AROUND` are now required to achieve a quiescent state. Because the signal relationships have been crystallised, there is no further need to use `finish_GAMMA`. All of the subsequent iterations of the simulation loop occur in 0-time. Since these steps are treated in an identical fashion to their unit-delay counterparts, there should be no surprise at the result:

$$
\begin{array}{l}
\vdash \forall now\ a\ b\ c\ beh.\\
\quad \texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ nandgte\ impl}\ (\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\})\ (\mathsf{signal}\ \{\mathrm{TMP}\})\ \ldots\right)\ \texttt{in}\\
\quad \texttt{let}\ \mu = \left(\mathsf{get_delays_cs}\ \Pi, now, \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c),(\mathrm{TMP},\neg(a\wedge b))\},\ldots\right)\ \texttt{in}\\
\qquad \mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh =\\
\left(\begin{array}{l}
beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a\wedge b)),(\mathrm{TMP},(a\wedge b))\}\\
\qquad\qquad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c),(\mathrm{TMP},\neg(a\wedge b))\}\\
\qquad\qquad \texttt{else}\ \{\}
\end{array}\right)
\end{array}
$$

The repeated iterations around the simulation loop at time $now+1$ left nested calls to $\mathsf{add_to_}\theta$ with identical time arguments in the trace of past behaviour. These were reduced via Theorem 4.3.5.3 when the simulation θ was transformed into the behaviour at quiescence through the use of rule sim_1. The remaining instance of $\mathsf{add_to_}\theta$ at time $now+1$ resulted in the first branch of the conditional seen in the derived behaviour. The simulator has therefore provided a way of abstracting from multiple δ-steps into the world of macro time simply through its operation.

The signal TMP also has important implications. Recall that the value associated with the signal in the original term has changed from *tmp* to $\neg(a\wedge b)$ as a result of its specification during the derivation. So, without actually knowing the initialisation conditions for the signal, the steps in the derivation allowed them to be determined. This was an instance of human intervention after an application of

program at quiescence with the conditionals that they represent. The result of the simplification is:

$$
\begin{array}{l}
\vdash \forall now\ a\ b\ c\ beh. \\
\quad \texttt{let}\ \Pi = \mathsf{dflat}\left(\mathsf{arch\ nandgte\ spec}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\ldots\left(\{\mathrm{A},\mathrm{B}\} : \mathrm{C} \overset{\mathsf{inert}}{:=} (\mathrm{A}\ \mathsf{nand}\ \mathrm{B},1)\right)\right)\texttt{in} \\
\quad \texttt{let}\ \mu = \left(\mathsf{get_delays_cs}\ \Pi, now, \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\},\{\mathrm{A}\},\lambda t.(\{\},\{\})\right)\texttt{in} \\
\quad\quad \mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh = \\
\left(\begin{array}{l} beh = \lambda t.\texttt{if}\ (t \geq now+1)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a \wedge b))\} \\ \qquad\quad \texttt{elseif}\ (t \geq now)\ \texttt{then}\ \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c)\} \\ \qquad\quad \texttt{else}\ \{\} \end{array}\right)
\end{array}
$$

It is important to note that the use of add_to_θ has transformed the discrete nature of time used by the simulator into continuous ranges more amenable to human reasoning. Strictly speaking, $t \geq now$ could be replaced by $t = now$ as there are no intervening units of time between *now* and $now+1$. Additionally, the `if-then` construction has replaced the HOL conditional here to aid legibility. The ellipsis (…) indicates where the local signal declarations should go. There are none, and they have been omitted for purposes of conciseness.

6.1.2 Behaviour of the Implementation

A similar sequence of steps may be carried out in the derivation of a behaviour for the implementation of the NAND gate. The first is to again start with a term which describes the architecture in question, along with its starting environment.

$$
\begin{array}{l}
\texttt{"let}\ \Pi = \mathsf{dflat}\left(\begin{array}{l} \mathsf{arch\ nandgte\ impl}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\left(\mathsf{signal}\ \{\mathrm{TMP}\}\right) \\ \qquad \left(\left(\{\mathrm{A},\mathrm{B}\} : \mathrm{TMP} \overset{\mathsf{inert}}{:=} (\mathrm{A}\ \mathsf{and}\ \mathrm{B},1)\right) || \left(\{\mathrm{TMP}\} : \mathrm{C} \overset{\mathsf{inert}}{:=} (\mathsf{not}\ \mathrm{TMP},0)\right)\right) \end{array}\right)\texttt{in} \\
\texttt{let}\ \mu = \left(\mathsf{get_delays_cs}\ \Pi, now, \{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},c),(\mathrm{TMP},tmp)\},\{\mathrm{A}\},\lambda t.(\{\},\{\})\right)\texttt{in} \\
\quad \mu \vdash \langle \Pi, \lambda t.\{\}\rangle \xrightarrow{sim} beh\texttt{"}
\end{array}
$$

As seen earlier, the architecture has become a bit more complex. The interest here is to see how the δ-step introduced by the 0-delay signal assignment gets treated during the derivation.

The same steps employed in arriving at a behaviour for the specification are also used in the case of the implementation. The difference is what happens at time

The result is that the starting statement is converted into an equality between it and the simulation behaviour at the next point of computation, $now+1$. The new state of the signals is $\{(\mathrm{A},a),(\mathrm{B},b),(\mathrm{C},\neg(a\wedge b))\}$. The problem is that NEXT_GAMMA_CONV could not ascertain what the new events (if any) should be. Clearly the values of A and B have not changed. The question is whether or not the new value of C, $\neg(a\wedge b)$, is a change from its old one, c. There is obviously not enough contextual information about the relationship between these two values to make a judgement.

As a result, the conversion finish_GAMMA must now be used to clear up the ambiguity by inserting the appropriate case split into the derived behaviour. Intuitively, the existence of such an event really should not matter. The signal C is not hooked up to another device, and cannot cause a ripple-through effect. The execution of the program should therefore be the same irrespective of the presence or non-presence of an event on C at time $now+1$. The following application inserts the necessary conditional:

```
let t2 = RIGHT_CONV_RULE finish_GAMMA t1
```

Recall the discussion of conversions in Chapter 5, and the fact that they result in a theorem about the logical equivalence of one term with respect to another. RIGHT_CONV_RULE is a function which applies a conversion to the right hand side of such an equality. In the context of the current example, it is applying finish_GAMMA to the just-derived simulation behaviour at time $now+1$.

Knowledge of the simulation loop leads to the understanding that the introduction of an event on C will eventually filter through the system and not generate any associated activity. Since the signal values are now represented by relationships of the initial inputs, finish_GAMMA will not be needed in subsequent iterations of the simulation loop. The job of arriving at a quiescent statement of behaviour may be done simply by repeated applications of ONCE_AROUND.

A quiescent simulation leaves a theorem that is almost in the desired form. It is a statement of equality between the initial simulation specification and the behaviour of that simulation. The behaviour does however contain two parts that need to be dealt with. The first is the conditional branch introduced by finish_GAMMA. The behaviour after time $now+1$ is the same irrespective of the value of C. This allows the branch to be simplified down to one instance of that behaviour. The other simplification is to replace calls to add_to_θ which had accumulated in the simulation θ and had subsequently become the behaviour of the

entities, architectures and signals will be given in roman font. Constants relating to the abstract syntax will be given in sans serif. Variables will be shown in *math italic*.

6.1.1 Behaviour of the Specification

Due to the deep embedding approach, it is possible to set up a term in the HOL system that characterises the semantics of the specification in the following way:

```
"let Π = dflat (arch nandgte spec ({A,B},{C},{}) (signal {}) ({A,B} : C :=inert (A nand B,1))) in
 let μ = (get_delays_cs Π, now, {(A,a),(B,b),(C,c)}, {A}, λt.({},{})) in
     μ ⊢ ⟨Π, λt.{}⟩ —sim→ beh"
```

Those parts of the term in `typewriter` font represent ML text. Note also the use of `let`-expressions in the term. Despite the fact that use is being made of abstract syntax, the terms of the embedded system can become *very* large. Term size management is therefore essential, and `let`-expressions provide the necessary organisation by minimising the amount of duplication within any given term.

The process of deriving a behaviour for the above term is accomplished by animating it with the rules of the semantics. Essentially, this is a symbolic simulation – "symbolic" in the sense that the values associated with each signal are logical variables, as in the state component of μ above. The conversions described in Chapter 5 are employed in this simulation, and are used to transform the term from a statement about the semantics of the simulation of a particular program into one of its behaviour over time. The result of the process will be a theorem about the simulation and its semantic equality to the behaviour represented by the variable *beh*. The actual derivation of this general behaviour could be a completely automatic process, but is broken down here into its separate components to aid in understanding the individual steps.

Conceptually, the derivation is nothing more than the execution of the program by the simulation engine as it is specified by the rules of the semantics. The first move therefore is to use `ONCE_AROUND` to perform a simulation cycle to determine where the next point of computation lies, move to it and figure out what the new environment there should be. These actions are accomplished by the following simple call to `ONCE_AROUND` where `tm` is bound to the above term:

```
let t1 = ONCE_AROUND tm
```

methodology of using the embedded semantics. The implementation will then employ of the same methodology in a similar derivation. The actual HOL session from which this study is extracted appears as Appendix B.

The two designs in question are of course instances of the same entity. The Femto-VHDL for them is:

```
                        entity nandgte is
                          port (A,B : in boolean;
                                C : out boolean);
                        end nandgte;

architecture spec of nandgte is   architecture impl of nandgte is
begin                               signal TMP : boolean;
  process (A,B)                   begin
  begin                             process (A,B)
    c <= A nand B after 1 ns;       begin
  end process;                        TMP <= A and B after 1 ns;
end spec;                           end process;

                                    process (TMP)
                                    begin
                                      C <= not TMP after 0 ns;
                                    end process;
                                  end impl;
```

Inertial delay is used here and in the other studies as it encompasses both the pre-emptive scheduling of transport delay as well as the set-up and hold constraints of inertial delay proper. The architecture for the implementation differs from the specification in that the monolithic NAND operation has been replaced by two parallel processes. A local signal has also been introduced. The first process assigns to this signal, which in turn assigns the result to the output.

In the interests of brevity, the rest of this study as well as those which follow it will revert to the use of abstract syntax for displaying Femto-VHDL text. The translations of the two programs just given are:

$$\mathsf{dflat}\left(\mathsf{arch\ nandgte\ spec}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\left(\mathsf{signal}\ \{\}\right)\left(\{\mathrm{A},\mathrm{B}\}:\mathrm{C}\overset{\mathsf{inert}}{:=}\left(\mathrm{A}\ \mathsf{nand}\ \mathrm{B},1\right)\right)\right)$$

$$\mathsf{dflat}\left(\begin{array}{l}\mathsf{arch\ nandgte\ impl}\left(\{\mathrm{A},\mathrm{B}\},\{\mathrm{C}\},\{\}\right)\left(\mathsf{signal}\ \{\mathrm{TMP}\}\right)\\ \quad\left(\left(\{\mathrm{A},\mathrm{B}\}:\mathrm{TMP}\overset{\mathsf{inert}}{:=}\left(\mathrm{A}\ \mathsf{and}\ \mathrm{B},1\right)\right)\|\left(\{\mathrm{TMP}\}:\mathrm{C}\overset{\mathsf{inert}}{:=}\left(\mathsf{not}\ \mathrm{TMP},0\right)\right)\right)\end{array}\right)$$

for the specification and implementation respectively. The above pattern of typefaces will also be followed throughout. Specifically, names of Femto-VHDL

CHAPTER 6

CASE STUDIES

Several case studies that make use of the formalisation that has been developed are now presented. While the studies do not involve terribly complex programs, they are sufficient for giving a flavour of the kind of reasoning that may be performed. The exposition begins with the simple example of a NAND gate, and progresses through to a more complex device.

As alluded to in Chapter 1, the studies will follow a general pattern:

1. Specify a high-level (i.e. algorithmic) representation of the device.
2. Derive, using the semantics, a general behaviour for the specification.
3. Specify a low-level implementation of the device.
4. Derive a general behaviour for the implementation in the same fashion as 2.
5. Prove that the two behaviours are the same.

The framework for each example will be the same, but different features of the semantics are illustrated by them. The first one concerns itself with demonstrating 0-delay signal assignments. The second deals not only with the equivalence of two components, but also with their provable difference to similar ones. The third example shows how one contends with initialisation and quiescence restrictions. The last one illustrates the use of a state-holding device.

6.1 A NAND Gate

The initial example is not only of use in introducing the way in which one works with the embedded semantics, but is also a demonstration of how 0-delay signal assignments are dealt with. In performing this example, the equivalence of two different versions of a NAND gate will be demonstrated. The first can be considered a "specification" of the behaviour of the gate, and makes use of the VHDL primitive Boolean operator `nand`. The second is an "implementation" that chains together an AND gate and an inverter. The specification will be used to illustrate the