Programming Research Group

A CALCULUS OF SIGNALS

C. Ratzko and J.W.Sanders

PRG-TR-06-00

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

A Calculus of Signals

C. Ratzko and J. W. Sanders

Abstract

An elementary theory is proposed for reasoning about circuits at the timed level. Its relationship to traditional differential calculus is explored; it is applied to the analysis of circuits with feedback; and algorithms for detection of transients and hazards are given. Finally a small case study, of a positive-edge-triggered register, is presented. Emphasis is on the use of laws which draw on the intuition gained from traditional differential calculus.

Key words: formal methods; signal; hazard analysis; timing.

1 Introduction

Ever since the switching model of Claude Shannon [27, 28], Boolean algebra has been effective in analysing the behaviour of circuits. Part of the reason for its persistence over 50 or 60 years is perhaps the coincidence that current VLSI technologies are based on products which act, at least at one level of abstraction, like switches. Nowadays there are powerful and subtle tools available for circuit simulation which enable the designer to synthesise and animate designs. But there seems to be a place for an intermediate-level formalism, preserving the simplicity of Shannon's static model but more realistic with respect to time than it and many of the various tools.

No doubt one of the reasons for the absence of such a model, at least in the case of real time domain, has been the complicated machinery underpinning differentiation and integration of non-smooth functions. The work of Laurent Schwartz [26] employing functional analysis to lay a foundation for Oliver Heaviside's calculus and George Temple's version using generalised functions [30] does not seem to have provoked a discrete simplification. The current work is an initial attempt to 'return to basics' to do just that. It uses none of those theories and no more than rudimentary (applied) logic. Indeed because of idempotence of conjunction and disjunction the theory of partial integration on which Schwartz's theory is based trivialises completely.

Many of the established models for hardware design include timing; for a typical cross section we refer to RUBY [17], CIRCAL [23], HOL [5, 21], and various other languages [7, 8, 14]. Several of the commercial systems have semantic models; important examples are VHDL [3, 19] and Verilog [15, 11, 31, 1, 32, 24]. There is also an established history of the use of temporal logic to study the timing aspects of hardware, for instance [10, 2, 22, 29, 20], and process algebra has been used in both the synchronous [16] and delay-insensitive [18] cases. More recently discrete duration calculus has also been used [34, 12, 13, 9]. Naturally there is a trade-off between expressivity of the model and algorithmic complexity of simulation based on it. For instance the state-of-the-art model [6] is typical in the way it abstracts time; by considering instead 13 types of transition it is able to provide a polynomial algorithm for min-max calculation. The approach developed here differs from those by being based on the idea of pointwise differentiation and integration of Boolean-valued signals to discuss signal rise and fall. An early version appeared in [25].

In this paper we present an elementary calculus for reasoning about signals. It may be thought of as a discrete version of distribution theory (in the sense of Schwartz) with potential application to the semantics of the languages above: at least at one level of abstraction. Any such theory must make firm the intuition that the Dirac function is the derivative of the Heavisde function; the proposed theory does so effortlessly, requiring no theory beyond Boolean algebra. It has not (yet) been used to provide the semantics of any particular language nor has it been implemented. Although it could have been presented, with some modifications, in a continuous form (i.e. with the real numbers for time) we have here chosen to present a discrete version.

The literatures on models for synchronous and asynchronous circuits seem to be almost disjoint, with the latter enjoying a renaissance in the past decade. Although the techniques for those two types of circuit appear quite different, they share a common foundation at the level of signal-per-time unit. It is this level at which our model is defined. It is comprehensive enough to reason about combinational features like race hazards and sequential features like feedback; whilst the fundamental asynchronous *events* and *transitions* are derived concepts. The result does not seem to be much more complicated than Boolean algebra. We hope the extra complication is in part allayed by the fact that the laws are strongly motivated by intuition from standard differential calculus; and we hope that feature makes it almost as easy to use as Boolean algebra is for static circuit analysis.

We use the following notation. The logical connectives are written: $\wedge$ for conjunction; $\vee$ for disjunction; $\leftrightarrow$ for equivalence; $\oplus$ for exclusive-or; $\neg$ for negation; 1 for *true* and 0 for *false*. A predicate is viewed as a Boolean-valued function. In chained reasoning we use the infix symbols $\equiv$ and $\Rrightarrow$ respectively for equivalence and implication. Functional application of function f to argument x is written $f.x$; however when the function is viewed as a predicate we use standard notation to exhibits its arguments (i.e. free variables). The type of all functions from A to B is written $A \to B$.

2 Signals

In this section we justify our choice of time domain, define signals, lift the operators of Boolean algebra so that they operate on signals, and define the delay operation on signals.

2.1 Time

A digital circuit is modelled using Boolean logic by naming each signal; by supposing each signal to assume, in the steady state, a Boolean value; and by expressing the relationship between the named signals as a formula of (predicate) logic. The laws of logic are then used to reason about the circuit: to verify, analyse or synthesise it. Whilst it is for many purposes appropriate to assume stabilisation, it is inappropriate in the analysis of time-dependent circuit features like race hazards.

We suppose that, for the purpose of reasoning about such phenomena, time is modelled by the integers $\mathbb{Z}$ (positive and negative). A more precise choice would have been the domain of rational numbers, which has the property that between any two times lies an intermediate time. An even more precise choice would have been the domain of real numbers, which is closed under limits. However by assuming that there is some smallest positive time ε we are able to reason about the phenomena in which we are interested whilst remaining within the time domain which resembles the set of multiples of ε which, to within a change of variable, can be thought of as the integers.

We follow the convention of relativity theory and deem that the negative integers represent time past, 0 represents the present and the positive integers represent the future.

Were the natural numbers to be chosen instead then each signal would have to be intitialised nondeterministically at time 0 (or else predicates would have to be qualified by a 'boundary condition' equal to the value of the signal at time 0). By choosing instead $\mathbb{Z}$ we defer that problem to the point at which each time-recursive equation has two solutions.

2.2 Signals

A signal is a Boolean valued time-dependent function. Writing $\mathbb{B}$ for the type $\{0, 1\}$ of Booleans, the type of all signals is thus defined to consist of all functions from integers to Booleans

$$\mathbb{S} \mathrel{\hat{=}} \mathbb{Z} \to \mathbb{B}.$$

For example the Heaviside function is a signal which was low in the past, goes high now and will stay high in future:

$$\begin{aligned}&\mathit{heavi} : \mathbb{S}\\ &\mathit{heavi}.t \;=\; (t \geq 0)\,.\end{aligned}$$

The Dirac delta function is a signal which is high now but was low in the past and will be low in future:

$$\begin{aligned}&\delta : \mathbb{S}\\ &\delta.t \;=\; (t = 0)\,.\end{aligned}$$

The intuition that δ is the derivative of *heavi* will be formalised in section 3. A clock having period 2, which is high now and which alternates is a signal:

$$\begin{aligned}&\mathit{clock} : \mathbb{S}\\ &\mathit{clock}.t \;=\; \neg(t \;\mathit{mod}\; 2)\,,\end{aligned}$$

where $(t \;\mathit{mod}\; 2)$ denotes the integer remainder of t after division by 2.

2.3 Boolean operations

The calculus of signals to be presented extends Boolean algebra in the following way. If $\otimes$ is a binary Boolean operator then $\otimes$ is lifted pointwise to signals:

$$\begin{aligned}&\otimes : (\mathbb{S} \times \mathbb{S}) \to \mathbb{S}\\ &(s \otimes s').t \;=\; s.t \otimes s'.t\,.\end{aligned}$$

Similarly for the unary operator of negation and the nullary operators, namely the constants $\mathbf{0}$ and $\mathbf{1}$.

A consequence of lifting is that the signal calculus inherits the laws of Boolean algebra. Signals s and s' are equal as functions iff they are pointwise equivalent

$$s = s' \;\equiv\; \forall t{:}\mathbb{Z} \cdot s.t \leftrightarrow s'.t\,.$$

For $s{:}\mathbb{S}$, where convenient we use Dijkstra's notation $[\,s\,]$ iff $\forall t{:}\mathbb{Z} \cdot s.t$; in that notation the previous equivalence becomes

$$s = s' \;\equiv\; [\,s \leftrightarrow s'\,]\,.$$

2.4 Delay

The previous signal operators produce output instantaneously by responding immediately to input. To introduce a time delay we define the retiming operator d which takes an integer n and a signal s and returns a signal which at time t behaves like s at time $t-n$. To express it we write the first argument, the delay, as a subscript; as usual functional application binds to the left.

$$\begin{aligned} & d : \mathbb{Z} \to \mathbb{S} \to \mathbb{S} \\ & d_n.s.t \;\hat{=}\; s.(t-n)\,, \end{aligned}$$

We abbreviate the delay function with unit delay, d_1, to d. Thus $d_n.s = d^n.s$. Also we adopt the convention that application of the delay function binds more tightly than that of the lifted logical operators.

It is routine to check that the delay operator d distributes over the unary and binary signal operators defined in the previous section; for example:

$$d_n.(s \otimes s') \;=\; d_n.s \otimes d_n.s'\,.$$

Several laws follow from the laws of arithmetic on $\mathbb{Z}$: successive delays are the same as one compound delay; successive delays commute; and a delay by zero is no delay:

$$\begin{aligned} d_n.(d_m.s) &\;=\; d_{n+m}.s \\ d_n.(d_m.s) &\;=\; d_m.(d_n.s) \\ d_0.s &\;=\; s\,. \end{aligned}$$

Finally the delay operator satisfies an adjointness law:

$$(d_n.s = s') \;\equiv\; (s = d_{-n}.s')\,.$$

As trivial application of those laws follow simple facts like

$$d.clock \;=\; \neg clock\,.$$

In section 5 we return to the equation $(d.s = \neg s)$ and show that its only solutions are *clock* and $\neg$*clock*.

2.5 Timed gates

A stable gate is modelled, in Boolean algebra, by a predicate whose vector of free variables is a tuple of Booleans. For example an *and* gate with inputs x and y and output z is modelled

$$\begin{aligned} & and : (\mathbb{B} \times \mathbb{B} \times \mathbb{B}) \to \mathbb{B} \\ & and(x,y,z) \;=\; z \leftrightarrow (x \wedge y)\,. \end{aligned}$$

Using the delay operator d we can define a *timed* version with response delay n:

$$\begin{aligned} & tand : (\mathbb{S} \times \mathbb{S} \times \mathbb{Z} \times \mathbb{S}) \to \mathbb{S} \\ & tand(x,y,n,z) \;=\; z \leftrightarrow d_n.(x \wedge y)\,. \end{aligned}$$

Similarly is defined an arbitrary timed gate corresponding to arbitrary operator $\otimes$.

A *timed buffer* simply interposes a delay between input x and output z:

$$\begin{aligned} & tbuff : (\mathbb{S} \times \mathbb{Z} \times \mathbb{S}) \to \mathbb{S} \\ & tbuff(x,n,z) \;=\; z \leftrightarrow d_n.x\,. \end{aligned}$$

To 'retrieve' the delay of a timed gate $\otimes(x,y,n,z)$ from its specification we write:

$$n = delay.\otimes\,.$$

2.6 Circuit delay

Suppose a combinational circuit C has input vector x and single output z which is the (direct) result of combining the outputs d and e of two subcircuits D and E by gate $\otimes(d,e,z)$. Suppose that all wires have delay 0; for if they are long or capacitive enough to impose a delay then we assume them to be modelled by a timed buffer.

The maximum delay *dmax* of the timed circuit C is calculated recursively from the delays of circuits D and E and gate $\otimes$:

$$dmax.C \;=\; delay.\otimes \;+\; (dmax.D \sqcup dmax.E)$$

where $\sqcup$ denotes infix maximum. A similar formula holds for the minimum delay. The minimum and maximum circuit delays reflect the fact that a change to one of the inputs x may affect the value of the output signal z no earlier than after time *dmin.C* and no later than after time *dmax.C*. Note that the calculated extrema may be pessimistic as estimates of the actual extrema (see [4], section 1.2).

It is now possible to distinguish combinational circuits having the same functional specification but different timing behaviours. For example these two implementations of exor assume the delay of an *and* gate is a, that of an *or* gate is o and that of an *inverter* is i:

$$\begin{aligned} texor0 \;&=\; z \leftrightarrow d_a.(d_o.(s \vee s') \\ &\qquad\qquad\qquad \wedge \\ &\qquad\qquad\qquad d_i.(\neg d_a.(s \wedge s'))) \\ texor1 \;&=\; z \leftrightarrow d_i.(d_a.(d_i.(\neg s \wedge s') \\ &\qquad\qquad\qquad \vee \\ &\qquad\qquad\qquad d_a.(s \wedge d_i.(\neg s'))))\,. \end{aligned}$$

By standard logic *texor0* and *texor1* have the same untimed behaviour. But assuming $i = o = 1$ and $a = 2$, we find from the formulae above

$$\begin{aligned} & dmin.texor0 = 3\,, \qquad dmin.texor1 = 3\,, \\ & dmax.texor0 = 5\,, \qquad dmax.texor1 = 4\,. \end{aligned}$$

Whilst they have the same minimum delay, the second takes less time to stabilise in the worst case.

3 Differentiation

In reasoning with signals one is typically interested in not just their values but whether they rise or fall—in their (discrete) 'slope'. In this section, motivated by traditional differential calculus, we define the discrete rising, falling and combined derivatives of a signal and investigate the resulting calculus.

3.1 One-sided differentiation

The rising, or positive, derivative of a signal s is defined to be a signal which is high at time t iff s was low at time $t-1$ and high at time t:

$$\begin{aligned} & D_+ : \mathbb{S} \to \mathbb{S} \\ & D_+.s \ = \ \neg d.s \wedge s \,. \end{aligned}$$

Similarly the falling, or negative, derivative is:

$$\begin{aligned} & D_- : \mathbb{S} \to \mathbb{S} \\ & D_-.s \ = \ d.s \wedge \neg s \,. \end{aligned}$$

Evidently D_+ and D_- are well defined. We adopt the convention that application of the differentiation operators binds more tightly than that of the lifted logical operators.

The laws for differentiation, which follow by routine calculation, are summarised in the following theorem. The first pair of laws connects the two kinds of differentiation; the second gives the discrete version of the product law; the third is the analogue of the product law for disjunction; and the fourth, more involved, pair provide the analogue for exor.

Theorem 1 For any signals s and s',

1. $$\begin{aligned} D_+.\neg s &= D_-.s \\ D_-.\neg s &= D_+.s \end{aligned}$$

2. $$\begin{aligned} D_+.(s \wedge s') &= (D_+.s \wedge s') \vee (s \wedge D_+.s') \\ D_-.(s \wedge s') &= (D_-.s \wedge d.s') \vee (d.s \wedge D_-.s') \end{aligned}$$

3. $$\begin{aligned} D_+.(s \vee s') &= (D_+.s \wedge \neg d.s') \vee (\neg d.s \wedge D_+.s') \\ D_-.(s \vee s') &= (D_-.s \wedge \neg s') \vee (\neg s \wedge D_-.s') \end{aligned}$$

4. $$\begin{aligned} D_+.(s \oplus s') \ = \ & (D_+.s \wedge \neg s' \wedge \neg d.s') \vee \\ & (D_-.s \wedge s' \wedge d.s') \vee \\ & (\neg s \wedge \neg d.s \wedge D_+.s') \vee \\ & (s \wedge d.s \wedge D_-.s') \\ D_-.(s \oplus s') \ = \ & (D_-.s \wedge \neg d.s' \wedge \neg s') \vee \\ & (D_+.s \wedge d.s' \wedge s') \vee \\ & (s \wedge d.s \wedge D_+.s') \vee \\ & (\neg s \wedge \neg d.s \wedge D_-.s') \,. \end{aligned}$$

A straightforward corollary of that result formalises a fact we expect of any (Boolean) signal: it does not rise [fall] on consecutive intervals.

Corollary 2 For any signal s,

$$\begin{aligned} D_+.s \wedge D_+.d.s &= \mathbf{0} \\ D_-.s \wedge D_-.d.s &= \mathbf{0} \,. \end{aligned}$$

It might be thought possible, due to the 'product' law, to derive a rule for integrating signals 'by parts'. We shall later see that is not the case. Another striking difference with the standard theory of differentiation is that in the discrete case, by idempotence, higher derivatives simplify immediately.

Theorem 3 For any signal s and positive integer m,

$$\begin{aligned} D_+^m.s &= D_+.s \\ D_-^m.s &= d_{m-1}.D_-.s \,. \end{aligned}$$

Since $D_+.s$ holds only when s rises and not when it falls again, equivalences of the form $(D_+.s \ = \ s')$ have an infinite number of solutions; similarly for $(D_-.s \ = \ s')$. Thus, for example, only when

$$D_+.s.t \ \wedge \ D_+.s.(t{+}2)$$

for some t can we deduce $D_+.s.(t{+}1)$. So prepared we can characterise the units of the differentiation operators:

Theorem 4 Signal s is a unit of D_+, that is $(D_+.s \ = \ s)$, iff it satisfies

$$s \wedge d.s \ = \ \mathbf{0} \,.$$

It is a unit of D_- iff it is the constant $\mathbf{0}$.

Proof: We reason as follows:

$$\begin{aligned} & D_+.s \ = \ s \\ \equiv \quad & \{\text{definition of } D_+\} \\ & \neg d.s \wedge s \ = \ s \\ \equiv \quad & \{\text{definition of } =, \leftrightarrow\} \\ & (\neg(\neg d.s \wedge s) \vee s) \wedge (\neg s \vee (\neg d.s \wedge s)) \\ \equiv \quad & \{\text{predicate calculus}\} \\ & \neg(s \wedge d.s) \,. \end{aligned}$$

♡

Note that the first equation is satisfied by infinitely many signals, all not high on two consecutive intervals; examples are $\mathbf{0}$ and the signals introduced earlier: δ, *clock* and $\neg$*clock*.

3.2 Two-sided differentiation

The (two-sided) derivative of a signal holds iff the signal either rises or falls at that time:

$$\begin{aligned} & D : \mathbb{S} \to \mathbb{S} \\ & D.s \ = \ D_+.s \vee D_-.s \,. \end{aligned}$$

It is simple to show that a signal does not rise and fall simultaneously. From that the next two equations result and provide alternative definitions of D ; the third states its time-invariance.

Theorem 5 For any signal s ,

$$\begin{aligned} D.s &= s \oplus d.s \\ D.s &= D_+.s \oplus D_-.s \\ D.(d^n.s) &= d^n.(D.s) . \end{aligned}$$

Proof: The first equivalence is established as follows:

$$\begin{aligned} & D_+.s \oplus D_-.s \\ \equiv & \qquad \text{definition of } \oplus \\ & (D_+.s \vee D_-.s) \wedge \neg(D_+.s \wedge D_-.s) \\ \equiv & \qquad \text{observation preceding theorem 5} \\ & (D_+.s \vee D_-.s) \wedge \mathbf{1} \\ \equiv & \qquad \text{definition of } D \\ & D.s . \end{aligned}$$

The second and third are simple verifications. ♡

In the following we use whichever of those forms is convenient.

The laws for the two-sided derivative of a Boolean combination of signals are proved like those for one-sided derivatives; the last, for the case of exor, provides a welcome simplification.

Theorem 6 For any signals s and s' ,

1. $D.\neg s = D.s$
2. $D.(s \wedge s') = (D_+.s \wedge s') \vee (s \wedge D_+.s') \vee (D_-.s \wedge d.s') \vee (d.s \wedge D_-.s')$
3. $D.(s \vee s') = (D_+.s \wedge \neg d.s') \vee (\neg d.s \wedge D_+.s') \vee (D_-.s \wedge \neg s') \vee (\neg s \wedge D_-.s')$
4. $D.(s \oplus s') = D.s \oplus D.s'$.

It is a consequence of the first law that an equation

$$D.s = s' ,$$

for any $s' : \mathbb{S}$, does not have a unique solution.

It can be shown that neither $D.(s \wedge s')$ nor $D.(s \vee s')$ can be expressed as a 'linear combination' of $D.s$ and $D.s'$; that is the reason a rule for integrating by parts cannot be formulated.

Higher-order two-sided derivatives are manipulated by the next law:

Theorem 7 For any signal s and positive integer n ,

$$D^n.s = s \oplus d_n.s .$$

The laws in this section suffice to show that the Dirac δ is the (two-sided) derivative of the Heaviside function *heavi* . Indeed:

$$\begin{aligned} D_+.heavi &= d.\neg heavi \wedge heavi \\ D_-.heavi &= d.heavi \wedge \neg heavi \end{aligned}$$

from which it follows that

$$\begin{aligned} \forall t:\mathbb{Z} \cdot D_+.heavi.t &\leftrightarrow (t = 0) \\ \forall t:\mathbb{Z} \cdot D_-.heavi.t &\leftrightarrow 0 \end{aligned}$$

and hence that

$$D.heavi.t = \delta.t$$

as required.

We find that D has, like D_- , a fixed point. Solution of the general equation ($D.s = s'$) is treated in section 5.

Theorem 8 Signal $\mathbf{0}$ is the unit of the two-sided derivative D .

4 Transients and hazards

We have seen that circuits equivalent at the untimed level need not be equivalent at the timed level, and we have seen how to quantify their timed delay. In this section we consider the behaviour of timed circuits, including stabilisation, and define transients and hazards. In both cases we indicate algorithms for their detection.

4.1 Transients

The circuit whose single output z is the disjunction of inputs x and delayed input y exhibits a temporary fluctuation, or transient, on z . It is called a blip and analysed using the signal calculus as follows.

Assume that the *or* gate has delay 0 (since it is the 'race' between its two inputs which is relevant here) and that input y is delayed (perhaps by interposition of a buffer) by 1. The circuit is modelled

$$z = x \vee d.y .$$

If, at time 0, x goes high and y goes low we find

$$z \leftrightarrow heavi \vee d.\neg heavi$$

which simplifies to

$$z.t \leftrightarrow (t < 0) \vee (1 \leq t) ,$$

the expected blip signal.

More generally a transient is a blip in signal value caused by delay when one or more inputs change simultaneously. The length of transient in a given circuit C

is bounded by the difference between the maximum and minimum circuit delays, i.e. $dmax.C - dmin.C$.

A combinational circuit with k distinct inputs may exhibit a transient under $4^k - 2^k$ conditions. Indeed each input can rise, fall, remain low or remain high; that yields 4^k possibilities; but of those the 2^k in which no input rises or falls yield no transient. It is straightforward to see that a 'binary tree' circuit with *exor* gates at the nodes and buffers at the 'leaves' (which calculates the parity of its input word) exhibits a number of transients exponential in the number of its inputs. Thus the previous estimate is in general asymptotically exact.

One way to analyse the conditions under which a given circuit exhibits transients is to solve the function which the circuit calculates for each of the $4^k - 2^k$ different combinations of input signals. Each takes $O(n \log n)$ steps. An alternative algorithm is given in section 4.3. But first we formalise the notion of a blip.

4.2 Blip

There are two blips, one rising and the other falling. The rising blip operator $\Uparrow$ [falling blip operator $\Downarrow$] takes a positive integer n , which determines the length of the blip, and a signal s and returns a signal which is high [low] iff s rises [falls] at time $t = -n$ then falls [rises] at time $t = 0$. As for delays, the delay time is expressed as a subscript:

$$\begin{aligned} &\Uparrow, \Downarrow : \ \mathbb{Z} \to \mathbb{S} \to \mathbb{S} \\ &\Uparrow_n.s \ = \ (d_n.D_+.s) \wedge D_-.s \\ &\Downarrow_n.s \ = \ (d_n.D_-.s) \wedge D_+.s \,. \end{aligned}$$

It is important to note that our definition does not require a blip to remain constant between going high [low] and low [high]. For example if $n = 3$ it admits as rising blips both the signals $t \in \{-3, -2, -1\}$ and $t \in \{-3, -1\}$. The reason for our choice is that with different delays on inputs it is this more complex kind of blip which occurs in practice.

The following theorem summarises the conditions under which a blip of unit length occurs for the basic logic gates. It is proved, as usual, by routine manipulation. In it we write $\Uparrow$ and $\Downarrow$ respectively for $\Uparrow_1$ and $\Downarrow_1$.

Theorem 9 For any signals s and s' ,

1. $$\begin{aligned} \Uparrow.\neg s &= \Downarrow.s \\ \Downarrow.\neg s &= \Uparrow.s \end{aligned}$$

2. $$\begin{aligned} \Uparrow.(s \wedge s') \ &= \ (\Uparrow.s \wedge d.s') \vee (d.D_+.s \wedge D_-.s') \vee \\ &\quad\ (D_-.s \wedge d.D_+.s') \vee (d.s \wedge \Uparrow.s') \\ \Downarrow.(s \wedge s') \ &= \ (\Downarrow.s \wedge s' \wedge d_2.s') \vee \\ &\quad\ (s \wedge d_2.s \wedge \Downarrow.s') \end{aligned}$$

3. $$\begin{aligned} \Uparrow.(s \vee s') \ &= \ (\Uparrow.s \wedge \neg s' \wedge \neg d_2.s') \ \vee \\ &\quad\ (\neg s \wedge \neg d_2.s \wedge \Uparrow.s') \\ \Downarrow.(s \vee s') \ &= \ (\Downarrow.s \wedge \neg d.s') \ \vee \\ &\quad\ (d.D_-.s \wedge D_+.s') \ \vee \\ &\quad\ (D_+.s \wedge d.D_-.s') \ \vee \\ &\quad\ (\neg d.s \wedge \Downarrow.s') \,. \end{aligned}$$

The laws for exor each involve eight disjuncts and so are omitted. Laws for blips of greater length are slightly more involved and so also omitted.

4.3 Transient detection

In this section we describe an algorithm for detecting transients in any combinational circuit with k distinct inputs. Described informally in four stages, the correctness of the underlying method is a result of the previous theory.

1. The circuit is pre-processed into a form in which all gates have at most two inputs but which exhibits the same timing behaviour as the original circuit. That is achieved by using the usual technique for staggering two-input gates to achieve a given k-input gate; but now the final gate is assigned delay equal to that of the original k-input gate; the other, staggered, gates have zero delay. Evidently there is more than one way of achieving such staggering; but that does not concern us here. We also assume that each complement gate (with an inverter bubble on the output) is replaced by the non-complemented gate followed by an inverter, again with the final (inverter) gate being assigned the delay of the original complemented gate.

   This stage requires $O(k^2)$ steps.

2. The pre-processed circuit is represented as an acyclic connected directed graph (a tree with multiple paths representing shared subcircuits) in which each node represents a gate, leaves represent inputs and the root represents output. Edges are directed from input towards output. In the following the only subgraphs we consider correspond to sub-tree-like subgraphs, extending from a node (against the directed edges) to the leaves. Each node has either one subgraph (inverters and buffers) or two subgraphs (all other gates, by step 1). For circuits in which there are two or more distinct paths from a given input to the output we shall use pointers and graph reduction to save duplication.

   The complexity of this stage depends on the data representation of the graph but for reasonable representations can be assumed to be polynomial in k .

3. Each subgraph is annotated with its minimum and maximum delay, as defined in section 2.6.

   By starting at the leaves and working towards the root this can be achieved in $O(k \log k)$ steps.

4. We assume an abstract graph data type which conceals implementation of graph reduction and similar details but which provides these functions:

   - `isleafnode(T)` which returns true iff subgraph `T` represents an input
   - `left(T)`, `right(T)` which return respectively the left and right subgraphs of `T` (undefined if `T` is a leaf) and return the same subgraph if only one is present.
   - `dmin(T)`, `dmax(T)` which return the minimum and maximum delays of the subgraph `T` (as defined in section 2.6) and return 0 if `T` is a leaf.
   - `gate(T)` which retrieves the type of the gate which combines the outputs from subgraphs of `T` to yield the output at the root of `T`.
   - `eliminate(T,e)` which, given graph `T` and an expression `e` for the conditions which cause `T` to exhibit a transient, returns a graph which is transient-free under those conditions but which otherwise behaves like `T`.
   - `lookup(g,e,[f])` which, given a gate `g` and either one (`e`) or two (`e`,`f`) expressions encoding a set of conditions which give rise to transients, returns a new expression encoding an extended set of conditions which take into account the gate `g`. This function computes the new expression by applying the laws for blip.
   - `dnf(e)` which, given an expression `e` (encoding the conditions under which `T` exhibits transients), returns an equivalent expression in disjunctive normal form. In the worst case this function is of exponential complexity.

   The algorithm in this stage is, on input `T`, given in figure 1.

The algorithm is of exponential complexity, as expected from section 4.1. However from function `analyse` we see that there are two kinds of circuit for which the algorithm will be substantially more efficient:

- regular circuits where almost all paths have the same or similar delay (for example logic arrays); for then path delays differ by say at most the delay of a single inverter;
- circuits with widely differing path delays (for example some random logic circuits); for then additional optimisation is possible by storing, instead of just the minimum and maximum delays, the set of all delays of all subgraphs and using those to ensure the algorithm does not waste time looking for transients of 'impossible' length for the given circuit.

```
{calculate an upper bound on the length of the transient}
n := dmax(T) - dmin(T) ;
{start by looking for transients of length 1}
i := 1 ;
{look for transients of length i}
while  (i <= n) do
{loop invariant: 1 ≤ i ≤ n+1 and  T contains no transients of length i−1}
       e := analyse(T,i) ;
       e' := dnf(e) ;
       T := eliminate(T,e') ;
       i := i+1
endwhile
{return the transient-free circuit}
return(T)
```

Figure 1: The algorithm. Function `analyse(T,i)` is given in figure 2.

4.4 Hazards

For many circuits the cost of eliminating transients by techniques more subtle than pipelining (which is not always feasible) is prohibitive. A useful alternative is to make it the responsibility of the environment to change at most one input at a time and to ensure that the circuit is transient-free under that stronger condition.

Transients in this simpler situation are called (race) hazards. A falling output blip in such a situation is called a *high* hazard; a rising output blip is called a *low* hazard.

An example is provided by an *or* gate whose inputs come from a single signal y forked and input once directly and once after inversion. Assuming that the inverter has unit delay but the *or* gate none, the circuit is modelled

$$z \;=\; y \vee d.y \;.$$

If input goes low at time 0 that yields

$$z \;=\; \neg heavi \vee d.heavi$$

which simplifies to

$$z.t \;\leftrightarrow\; (t < 0) \vee (1 \leq t)$$

```
{start with an empty set of conditions}
c := { } ;
{investigate the gate at the root of subgraph T}
g := gate(T) ;
{decide whether or not the recursion stops}
if not ( isleafnode(T)
         or dmax(T)-dmin(T)
         or ( (g <> not) and
              (dmax(left(T)) <
               i+dmin(right(T)))) or
              (dmax(right(T)) <
               i+dmin(left(T))))
   if g = inv
   c := c ∪ lookup(g,analyse(left(T),i)) ;
   else
   c := c ∪ lookup(g,analyse(left(T),i),
                   analyse(right(T),i))
   endif
endif
{return the expression for the conditions which give
rise to transients in T}
return(c)
```

Figure 2: Function `analyse(T,i)`.

which is a falling blip, or a high hazard.

The next result *characterises* the conditions for a race hazard to occur in a circuit composed entirely of inverters and 2-input *and*, *or* and *exor* gates. The proof is by structural induction on the circuit.

Theorem 10 The output signal z of a circuit composed entirely of inverters and 2-input *and*, *or* and *exor* gates exhibits a race hazard iff changing one input signal yields internal signals x and y from which z is calculated directly as follows

1. $(x = d_e.\neg s)$, $(y = d_f.s)$ and $(z = d_\otimes.(x \otimes y))$ where $e \neq f$ and s is some signal. In that case
   (a) if $e < f$ then z exhibits a low hazard if $\otimes = \wedge$ and s goes low and a high hazard if $\otimes = \vee$ and s goes high
   (b) if $e > f$ then z exhibits a low hazard if $\otimes = \wedge$ and s goes high and a high hazard if $\otimes = \vee$ and s goes low.
2. $[\,x \leftrightarrow d_e.s \;\vee\; x \leftrightarrow d_e.\neg s\,]$, $[\,y \leftrightarrow d_f.s \;\vee\; y \leftrightarrow d_f.\neg s\,]$ and $[\,z \leftrightarrow d_\oplus.(x \oplus y)\,]$ where $e \neq f$ and s is some signal. In that case eight conditions give rise to a hazard: four of them to a low and four of them to a high hazard; details are omitted.

Each hazard has length $|\,e - f\,|$.

The previous example is the case 2(b) with $e = 1$ and $f = 0$.

4.5 Hazard detection

As expected, hazard detection is more efficient than transient detection. The following algorithm uses theorem 10 to identify critical gates and then applies laws of blips to calculate hazard conditions.

1,2. As in the transient-detection algorithm.

3. Each subgraph T is annotated with a set of identifiers corresponding to all the inputs that lead to it (perhaps indirectly) and to each identifier is associated the set of delays after which the given input arrives at the gate `gate`(T). Separate identifiers are used for inverted and non-inverted inputs.

   By annotating the graph starting at the leaf nodes and working towards the root, this stage can be accomplished in a number of steps polynomial in the number of inputs.

4. The graph is scanned root down, maintaining in set H pointers to the subgraphs corresponding to *and* and *or* gates for which a signal s occurs both with delay a in the input set of its left subgraph and inverted with delay $b \neq a$ in the right subgraph; or *vice versa*. Also maintained in H are pointers to *exor* gates for which a signal s (inverted or otherwise) occurs with delay a in the input set of its left subgraph and (again, inverted or otherwise) with delay $b \neq a$ in the right subgraph.

   This stage takes time polynomial in the number of inputs.

5. For each subgraph in H the blip laws are used to calculate the conditions for that subgraph to exhibit a hazard of length $|\,b-a\,|$ where a and b are defined by the previous stage, and to augment the expression which encodes those conditions with further conditions (immediate from the blip laws) that allow the blip to ripple to the root of the graph. Finally the expression is converted to disjunctive normal form.

   This stage takes time exponential in the number of inputs.

5 Integration

In this section we consider the inverse of a two-sided discrete derivative. After defining the discrete integral we investigate its laws and illustrate their use in solving simple discrete differential equations.

5.1 Definition

In the equation $(D.s = s')$, signal s can be thought of as the integral of signal s'. It turns out that, for any given signal s', there are exactly two such solutions s, one the negation of the other. So although the integral operator ought properly to be defined as a relation on $\mathbb{S}$ we adopt the practice, by analogy with the indefinite integral in standard calculus, of writing it as a function to an equivalence class (containing just two elements) of signals. The analogy is all the closer since any signal s and its negation differ by only a constant; indeed

$$\begin{array}{rcl} s & = & s \oplus \mathbf{0} \\ \neg s & = & s \oplus \mathbf{1}\,. \end{array}$$

However before defining the integration operator we must establish the claim above about its solutions. We do so by showing first that there is at least one solution and second that there are in fact two.

Theorem 11 1. For given signal s' the equivalence $(D.s \;=\; s')$ has solution

$$s.t \;=\; \begin{cases} 0 & \text{if} \quad t = 0 \\ s.(t-1) \oplus s'.t & \text{if} \quad t > 0 \\ s.(t+1) \oplus s'.(t+1) & \text{if} \quad t < 0\,. \end{cases}$$

2. If signal s satisfies $(D.s \;=\; s')$ then also $\neg s$ satisfies it: $(D.\neg s \;=\; s')$. Furthermore s and $\neg s$ are the only solutions.

Proof:

1. Signal s is well defined since for all $t:\mathbb{Z}$ the value of $s.t$ can be evaluated, fanning out from the origin, by combining known values of $s'.t$ and $s.0 = 0$ using $\oplus$.

   The fact that it is a solution uses:

$$s' \in I.s \;\equiv\; s' \in \{s \oplus d.I.s \oplus \mathbf{c} \mid c:\mathbb{B}\} \tag{1}$$

   which is proved as follows.

$$\begin{array}{ll} \mathbf{1} & \\ \equiv & \text{theorem 5} \\ D.s \;=\; (s \oplus d.s) & \\ \equiv & \text{laws of } \oplus \\ (D.s \oplus d.s) \;=\; s & \\ \equiv & \text{property of } I \\ \exists c:\mathbb{B} \cdot I.(D.s \oplus d.s \oplus \mathbf{c}) \;=\; I.s & \\ \equiv & \text{laws of } I \text{ and constants} \\ \exists c:\mathbb{B} \cdot (I.D.s) \oplus (d.I.s) \oplus \mathbf{c} \;=\; I.s & \\ \equiv & \text{definition of } I \\ \exists c:\mathbb{B} \cdot s \oplus d.I.s \oplus \mathbf{c} \;=\; I.s\,. & \end{array}$$

   From that we deduce (1). Noting that

$$\begin{array}{ll} \forall t:\mathbb{Z} \cdot s.t \;=\; s.(t-1) \oplus s'.t & \\ \equiv & \text{changing variable } b := b{+}1\text{, laws of } \oplus \\ \forall t:\mathbb{Z} \cdot s.t \;=\; s.(t{+}1) \oplus s'.(t{+}1)\,, & \end{array}$$

   the defined signal s is now seen by (1) to be a solution to the differential equation.

2. The fact that the negation of a solution is again a solution follows from theorem 6. Suppose that a, $\neg a$ and b are solutions s of $(D.s \;=\; s')$ with b equal to neither a nor b. We shall reach a contradiction.

   Since each signal is Boolean valued,

$$\forall t:\mathbb{Z} \cdot (b.t = a.t) \vee (b.t = \neg a.t)\,. \tag{2}$$

   Since b, a and $\neg a$ are distinct

$$\exists t, t':\mathbb{Z} \cdot (b.t \neq a.t) \wedge (b.t' = a.t')\,. \tag{3}$$

   Thus from (2) and (3)

$$\begin{array}{l} \exists t:\mathbb{Z} \cdot \left(\begin{array}{l} b.t = a.t \\ b.(t{-}1) \neq a.(t{-}1) \end{array} \right) \\ \vee \\ \exists t:\mathbb{Z} \cdot \left(\begin{array}{l} b.t \neq a.t \\ b.(t{-}1) = a.(t{-}1) \end{array} \right)\,. \end{array}$$

   In the case of the first disjunct we find

$$\begin{array}{ll} D.b.t & \\ = & \text{theorem 5} \\ (b \oplus d.b).t & \\ = & \text{pointwise operations} \\ b.t \oplus d.b.t & \\ = & \text{definition of } d \\ b.t \oplus b.(t{-}1) & \\ \neq & \text{assumption} \\ a.t \oplus a.(t{-}1) & \\ = & \text{retracing the above with } a \text{ for } b \\ D.a.t\,. & \end{array}$$

   But that contradicts $(D.b = D.a)$. The second case is similar so the result follows. ♡

Write $\mathbb{S}/\sim$ for the space of equivalence classes of signals under the equivalence relation

$$s \sim s' \;=\; (s' = s) \vee (s' = \neg s)\,.$$

Then the integration operator I is defined, inverse to differentiation, as follows:

$$I : \mathbb{S} \to \mathbb{S}/\sim$$
$$s' \in I.s \quad \equiv \quad (D.s' = s)\,.$$

If $s' \in I.s$ we write $I.s = s' \oplus \mathbf{c}$ where $c:\mathbb{B}$ is called a constant of integration. That travesty of notation is justified, just as for the indefinite integral in standard calculus, by its convenience. However it means that we must be careful, when reasoning algebraically, in assigning such constants. For example to simplify the expression $I.s \vee I.s$ we reason:

$$(s \oplus \mathbf{c}) \vee (s \oplus \mathbf{c}')$$
$$\equiv \qquad \text{definition of } \oplus$$
$$(s \wedge \neg\mathbf{c}) \vee (\neg s \wedge \mathbf{c}) \vee (s \wedge \neg\mathbf{c}') \vee (\neg s \wedge \mathbf{c}')$$
$$\equiv \qquad \text{distributivity}$$
$$(s \wedge (\neg\mathbf{c} \vee \neg\mathbf{c}')) \vee (\neg s \wedge (\mathbf{c} \vee \mathbf{c}'))$$
$$\equiv \qquad \text{laws for manipulating constants}$$
$$(s \wedge \mathbf{c}'') \vee (\neg s \wedge \mathbf{c}''')\,.$$

Thus by choosing the appropriate number of constants we avoid fallacious conclusions (like that the term simplifies to $\mathbf{1}$).

Simple examples of integrals include the following.

1. $I.\mathbf{0} = \mathbf{0} \oplus \mathbf{c}$, which embraces both solutions $\mathbf{0}$ and $\mathbf{1}$.

   Iterating, the solutions of $I^2.\mathbf{0}$ are those of $I.\mathbf{1}$ and $I.\mathbf{0}$, which embrace $\mathbf{0}$, $\mathbf{1}$, and the clocks (defined in section 2.2) *clock* and $\neg$*clock* and no further solutions. More generally

   $$I^{n+1}.\mathbf{0} \;=\; I^n.\mathbf{c}\,.$$

2. The two solutions of the differential equation $D.s = \delta$ (for δ defined in section 2.2) are the signals *heavi* and $\neg$*heavi*.

3. The signal *clock* could have been defined by the differential equation

   $$D.\mathit{clock} \;=\; \mathbf{1}$$

   which is now seen to have solutions *clock* and $\neg$*clock*. Justification for that claim is conveniently phrased in terms of *clock* satisfying $d.s = \neg s$ and goes as follows:

   $$d.s = \neg s$$
   $$\equiv \qquad \text{definitions of } = \text{ and } \leftrightarrow$$
   $$(d.s \wedge \neg s) \vee (\neg d.s \wedge s) = \mathbf{1}$$
   $$\equiv \qquad \text{definitions of } D_+ \text{ and } D_-$$
   $$(D_-.s) \vee (D_+.s) = \mathbf{1}$$
   $$\equiv \qquad \text{definition of } D$$
   $$D.s = \mathbf{1}\,.$$

5.2 Laws

The first law is a summary:

Theorem 12 For any signals s and s',

$$s' \in I.(D.s) \quad \equiv \quad \exists c:\mathbb{B} \cdot (s' = s \oplus \mathbf{c})\,.$$

The second law establishes time invariance:

Theorem 13 For any signal s and positive integer n,

$$I.(d^n.s) \;=\; d^n.(I.s)\,.$$

It turns out that simplification of the integrals of $\neg s$, $s \wedge s'$ and $s \vee s'$ requires use of equivalence (1). However the case of *exor* is simpler:

Theorem 14 For signals s and s',

$$I.(s \oplus s') \;=\; I.s \oplus I.s'\,.$$

For higher-order integrals there is an extension of equivalence (1).

Theorem 15 For any nonconstant signal s and positive integer n,

$$I^n.x \;=\; s \oplus d^n.I^n.s \oplus I^n.\mathbf{0}\,.$$

Proof: We reason as one might in standard calculus:

$$D^n.x \;=\; s \oplus d^n.s$$
$$\equiv \qquad \text{integrate both sides}$$
$$I.(D^n.s) \;=\; I.(s \oplus d^n.s)$$
$$\equiv \qquad \text{theorems 14, 13}$$
$$I.(D^n.s) \;=\; I.s \oplus d^n.I.s$$
$$\equiv \qquad \text{theorem 12}$$
$$I.(D^{n-1}.s) \oplus \mathbf{c} \;=\; I.s \oplus d^n.I.s$$
$$\equiv \qquad \text{repeating } n{-}1 \text{ times}$$
$$s \oplus I^n.\mathbf{c} \;=\; I^n.s \oplus d^n.I^n.s$$
$$\equiv \qquad \text{Boolean algebra}$$
$$I^n.s \;=\; s \oplus d^n.I^n.s \oplus I^n.\mathbf{0}\,.$$

♡

Theorem 15 will be applied in the next section.

6 Feedback

So far, although we have studied the delay operator, we have considered only combinational circuits. In the remainder of the paper we investigate circuits containing feedback.

In this section we use the calculus to analyse a simple such circuit. We then extend the delay operator to a form to be used in the next section in analysing a more subtle example.

6.1 Example

The simplest circuit involving feedback consists of a single gate, say an *exor* gate with delay $n > 0$ one of whose inputs is fed back from output, z. It is modelled by

$$z \;=\; d_n.(x \oplus z)\,.$$

Using the calculus we reason

$$\begin{array}{l} z \;=\; d_n.(x \oplus z) \\ \equiv \hfill \text{distributivity of } d \\ z \;=\; (d_n.x) \oplus (d_n.z) \\ \equiv \hfill \text{laws of } \oplus \\ z \oplus (d_n.z) \;=\; (d_n.x) \\ \equiv \hfill \text{properties of } D \\ D^n.z \;=\; (d_n.x) \\ \equiv \hfill \text{definition of } I \\ D^{n-1}.z \;=\; (d_n.I.x \oplus \mathbf{c}) \\ \equiv \qquad \text{repeating } n{-}1 \text{ times, distributivity of } I \text{ over } \oplus \\ z \;=\; (d_n.I^n.x \oplus I^n.\mathbf{0})\,. \end{array}$$

We infer that, depending on initial conditions, the output z of even this very simple circuit can take one of 2^n forms. Hence in order to be able to express the signal z uniquely we must know its value on an interval of length n; for that permits determination of the n constants in the expansion of $I^n.\mathbf{0}$.

Unfortunately it is not always possible to resort to the power of integration to analyse feedback. For example were the *exor* gate replaced by an *or* gate

$$z \;=\; d_n.(x \vee z)\,,$$

since the $\oplus$ introduced by integrating does not distribute over $\vee$, we are unable to eliminate output z from the right-hand side. Indeed unless we know the form the input signal takes we can deduce little about the output of such a circuit other than $z \;=\; x \vee z$ which is immediate from the definition.

We consider a more interesting example of feedback in section 7. But first we require further notation for delays.

6.2 Delays revisited

The delay operator defined in section 2.4 can be viewed as a special case of a more general retiming operator r which we view as a curried function taking a function $f : \mathbb{Z} \to \mathbb{Z}$ and a signal s and returning a signal which at time t behaves like s at time $f.t$:

$$\begin{array}{l} r : (\mathbb{Z} \to \mathbb{Z}) \to \mathbb{S} \to \mathbb{S} \\ r_f.s \;=\; s \circ f\,. \end{array}$$

Thus $r_f.s.t = s.(f.t)$.

Laws for retiming are:

Theorem 16 If $f, g : \mathbb{Z} \to \mathbb{Z}$ and $s, s' : \mathbb{S}$ then

1. $r_f.(r_g.s) \;=\; r_{f \circ g}.s$
2. $r_f.(r_g.s) \;=\; r_g.(r_f.s)$ provided f and g commute
3. $(r_f.s \;=\; s') \;\equiv\; (s \;=\; r_{f^{-1}}.s')$ provided f is bijective.

In practice the first argument of r is monotonically increasing; otherwise it is infeasible or miraculous (in refinement-calculus terms). For example the function

$$\begin{array}{l} \mathit{flip} : \mathbb{Z} \to \mathbb{Z} \\ \mathit{flip}.t = -t \end{array}$$

reverses time. No circuit implements $s' \;=\; r_{\mathit{flip}}.s$, so in order to ensure that such a specification is feasible we restrict the first argument of r to be monotonically increasing:

$$t < t' \;\Rightarrow\; f.t \le f.t'\,.$$

Of course such functions are closed under composition and so the laws in the previous theorem persist with the assumption that f and g be monotonically increasing. (When they are not the resulting infeasible or miraculous specification is handled as in any refinement calculus.)

Observe that the delay operator d can be defined in terms of the retiming operator r by using the curried function

$$\begin{array}{l} \mathit{shift} : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z} \\ \mathit{shift}.n.t \;=\; t - n\,. \end{array}$$

For $r_{\mathit{shift}.n} \;=\; d_n$.

7 Case study

The retiming operator r just introduced enables us to analyse circuits which exhibit feedback but which would be difficult to reason about using the method of the previous section.

In this section we use the calculus to analyse the behaviour of a single-bit, positive-edge-triggered register

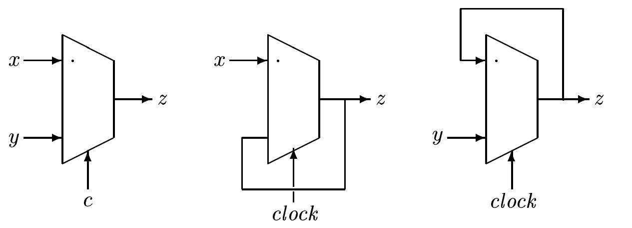

Figure 3: A 2-input multiplexor, a positive latch and a negative latch respectively.

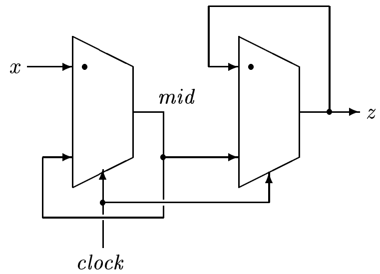

Figure 4: A positive-edge-triggered register constructed from a positive latch and a negative latch.

[33]. Suppose that we wish to construct it from the 2-input multiplexor depicted in figure 3.

The timed specification of the multiplexor is taken to be

$$z \;=\; d_s.((d_h.\neg c \wedge x) \vee (d_h.c \wedge y))$$

where s denotes the time it takes the selected input signal to reach the output and h denotes the time after which a change of c results in the alternative input being selected (though a further time s elapses before that change is reflected in the output z).

A multiplexor can be used to construct positive and negative latches (also in figure 3). Their timed specifications are

$$z \;=\; d_s.((d_h.\neg clock \wedge x) \;\vee\; (d_h.clock \wedge z)) \qquad (4)$$

$$z \;=\; d_s.((d_h.\neg clock \wedge z) \;\vee\; (d_h.clock \wedge y))\,. \qquad (5)$$

The control c has been relabelled *clock* to match the intended use of the circuit. Assume that the period of *clock* is T and that on every period *clock* is high for time TH and low for time $TL = T - TH$. For simplicity we shall assume that $h = 0$. By combining a postive latch with a negative latch as in figure 4 we obtain a postive edge-triggered register.

Its timed specification results from those of its ingredients by renaming the output of the positive latch to *mid*. However because of the two separate feedback loops it is not clear how to eliminate *mid*; thus

$$z \;=\; d_s.((\neg clock \wedge z) \vee (clock \wedge mid)) \qquad (6)$$

where

$$mid \;=\; d_s.((\neg clock \wedge x) \vee (clock \wedge mid))\,.$$

Fortunately an equivalent description, with no feedback, may be obtained using the retiming operator r. To motivate the choice of retiming function consider the behaviour of the negative latch. When the signal *clock* is high the negative latch acts like a buffer with delay s. When *clock* goes low the last value of input x that appeared on the output is stored in the feedback loop until *clock* goes high again. Thus we define curried function *rem*:

$$rem : \mathbb{S} \to \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$$

$$rem.s.n.t \;=\; \begin{cases} max\ U & \text{if } U \neq \{\,\} \\ t & \text{otherwise}, \end{cases}$$

where, as described above,

$$U = \{t' \in (t-n, t] \mid (D_+.s).t'\}\,.$$

Then we have:

Theorem 17 The specification (4) of the positive latch is equivalent to

$$z \;=\; d_s.r_{rem.clock.TL}.x$$

and that of the negative latch (5) to

$$z \;=\; d_s.r_{rem.\neg clock.TH}.x\,.$$

That yields, after some simplification, the expression we require for the positive-edge-triggered register:

Theorem 18 The specification (6) of the positive-edge-triggered register is equivalent to

$$z \;=\; d_{2s}.r_{rem.clock.T}.x\,.$$

8 Acknowledgements

The authors are grateful to Tony Hoare, Gordon Pace and Lance Watson for providing input in various ways.

References

[1] F. Balarin, Verilog HDL modelling styles for formal verification. In D. Agnew and L. Claesen, editors, *Computer Hardware Description Languages and their Applications (A-32)*. Elsevier Science Publishers BV (North-Holland), 1993.

[2] G. V. Bochmann, Hardware specification with temporal logic: An example. *IEEE Transactions on Computers*, C-31(3), March 1982.

[3] E. Börger, U. Glässer and W. Müller, Formal definition of an abstract VHDL '93 simulator by EA-machines. In C. Delgado Kloos and P. T. Breuer, editors, *Formal Semantics for VHDL*. Kluwer Academic Press, 1995.

[4] J. A. Brzozowski and C.-J. H. Seger, *Asynchronous Circuits*. Springer-Verlag, 1995.

[5] A. Camilleri, M. J.C. Gordon and T. Melham, Hardware verification using higher order logic. Technical report 91, University of Cambridge Computing Laboratory, 1986.

[6] S. Chakraborty, D. L. Dill and K. Y. Yun, Min-max timing analysis and an application to asynchronous circuits. Proceedings of the IEEE, **87**(2):332-346, 1999.

[7] C.-T. Chou, J.-L. Huang and M. Fujita, A high-level language for programming complex temporal behaviours and its translation into synchronous circuits, September 1996.

[8] Chrysalis Symbolic Design, Symbolic logic technology. January 1996. Available at `http://chrys.chrysalis.com/techinfo`.

[9] M. Fränzle, On appropriateness of the digitality abstraction taken in duration calculus to implementation development. ProCos document, Christian-Albrechts-Universität, 1995.

[10] R. Goldblatt, *Logics of Time and Computation*, chapter 6: Temporal logic. Number 7 in CLSI lecture notes. Center for the study of Language and Information, second edition, 1992.

[11] M. J.C. Gordon, The semantic challenge of Verilog HDL. In *Proceedings of the tenth annual IEEE symposium on Logic in Computer Science (LICS '95) San Diego*, June 1995.

[12] M. Hansen, C. Zhou and J. Staunstrup, A real-time duration semantics for citcuits. In *Tau '92: 1992 Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, ACM/SIDGA, September 1992.

[13] J. He and E.-R.,Olderog, From real-time specification to clocked circuit. *ProCos document, Department of Computer Science, Technical University of Denmark*, 1994.

[14] D. V. Hung and K. Kwang II, Verification via digitized models of real-time systems. UNU.IIST Research Report number 54, The United Nations University, 1996.

[15] IEEE, *Draft Standard Verilog HDL (IEEE 1364)*. 1995.

[16] inmos Limited, *Communicating Process Architecture*. Prentice-Hall, 1988.

[17] G. A. Jones and M. Sheeran, Circuit design in RUBY. In J. Staunstrup, editor, *Formal Methods for VLSI Design*, 13–70. Elsevier Science Publications BV, 1990.

[18] M. B. Josephs and J. T. Udding, An algebra for delay-insensitive circuits. Technical Report WUCS-89-54, Department of Computer Science, Washington University, March 1990.

[19] C. D. Kloos and P. T. Breuer, *Formal Semantics for VHDL*. Number 307 in The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1995.

[20] L. Lamport, The temporal logic of actions. *Technical report, Digital Equipment Corporation Systems Research centre*, 1993.

[21] T. F. Melham, *Higher-Order Logic and Hardware Verification*. Cambridge tracts in Theoretical Computer Science, **31**, 1993.

[22] B. Moszkowski, A temporal logic for multilevel reasoning about hardware. *Computer*, **2**(18):10–19, 1985.

[23] G. J. Milne, *Formal Specification and Verification of Digital Systems*. McGraw-Hill, 1994.

[24] G. J. Pace and J. He, Formal reasoning with Verilog HDL. In *Proceedings of the Workshop on Formal Techniques in Hardware and Hardware-like Systems*, Marstrand, Sweden, June 1998.

[25] J. W. Sanders, *Lectures on the Foundations of Hardware Design*. Course notes. Programming Research Group, OUCL, Oxford University, 1990.

[26] L. Schwartz, *Théorie des Distributions*, volumes 1 and 2, Hermann et Cie., Paris, 1950-1.

[27] C. E. Shannon, A symbolic analysis of relay and switching circuits. *Transactions of the AIEE*, **57**, 713–723, 1938.

[28] C. E. Shannon, The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, **28**, 59–98, 1949.

[29] T. G. Tang, A temporal logic language for hardware simulation, specification and verification. Technical report CMU-CS-88-194, Department of Computer Science, CMU, September 1988.

[30] G. Temple, Generalised functions. *Proc. Roy. Soc. A*, **228**, 175–190, 1955.

[31] D. E. Thomas, *The Verilog HDL, second edition.* Kluwer Academic, Boston, 1995.

[32] The Verilog Formal Equivalence (VFE) Project, Synthesizable Verilog: Syntax and semantics. 1997. Available at `http://www.cl.cam.ac.uk/users/djs1002/verilog.project`.

[33] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design.* 2nd edition, Addison-Wesley, 1992.

[34] C. Zhou, C. A. R. Hoare and A. P. Ravn, A calculus of durations. *Information Processing Letters*, **40**(5):269–276, 1991.

[35] D. Zissos, *Logic Design.* Academic Press, 1986.