Department of Computing Science and Mathematics
University of Stirling

Timed DILL: Digital Logic in LOTOS

Ji He and Kenneth J. Turner

Technical Report CSM-145

ISSN 1460-9673

April 1998

Department of Computing Science and Mathematics
University of Stirling

Timed DILL: Digital Logic in LOTOS

Ji He and Kenneth J. Turner

Department of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, Scotland

Telephone +44-786-467421, Facsimile +44-786-464551
Email jih@cs.stir.ac.uk, kjt@cs.stir.ac.uk

Technical Report CSM-145

ISSN 1460-9673

April 1998

Abstract

A timed extension to DILL (DIgital Logic in LOTOS) is discussed. The extension is based on ET-LOTOS (Enhanced Timed LOTOS) and is used to specify timed digital logic. The approach places timing properties in two categories: timing constraints and delays. A parallel-serial model is used to form a timed specification by composing the functionality of a digital component with timing constraints and delays. A variety of common timing constraints and delays have been specified, adding these as pseudo-components to the DILL library for subsequent reuse. The characteristics of other timed components in the DILL library are also discussed. The approach to timed specification and analysis is illustrated using a standard multiplexer component.

Keywords: Digital Logic, DILL (DIgital Logic in LOTOS), ET-LOTOS (Enhanced Timed LOTOS), HDL (Hardware Description Language), LOTOS (Language of Temporal Ordering Specification), TE-LOTOS (Time Extended LOTOS)

Contents

List of Figures

List of Tables

1 Introduction

In this paper, a timed extension is presented for DILL (DIgital Logic in LOTOS)[19], an approach and a language for specifying digital hardware designs in LOTOS. Extensions to the original version of DILL are described in [11].

Timing characteristics, such as the propagation delay of a logic gate, are an important property of digital components. They are critical for logic designs in two senses. First, the timing properties of each individual component in a logic circuit may determine if the circuit can function correctly. For example, potential race conditions and hazards in a digital circuit caused by the delays of components within a circuit could result in misoperation. Second, the timed characteristics of a digital circuit determines if the circuit satisfies requirements for timing performance and thus its usability in some circumstances. In recent years many hardware description languages, like VHDL [6] and Verilog [7], have been extended with more effective methods for handling timing. The corresponding languages VITAL (VHDL Initiative Towards ASIC Libraries [8]) and SDF (Standard Delay Format [9]) are becoming IEEE standards. For DILL it is also desirable to be able to specify timing properties so that the timed behaviour of logic designs can be analysed.

Untimed DILL is the previous work on DILL [11, 19]. It assumes that an event offer (which models the change in a digital signal) can happen at any time. Timed DILL is based on ET-LOTOS (Enhanced Timed LOTOS [13]) and is used to specify the behaviour in time of digital logic. ET-LOTOS was chosen as the foundation of Timed DILL mainly because ET-LOTOS is closely related to the timed semantics of the future ISO standard E-LOTOS (Enhancements to LOTOS [10]). It is hoped that Timed DILL will be easily transferable to E-LOTOS once the standard is mature. Because E(T)-LOTOS tools are currently under development, the authors have used TE-LOTOS (Time Extended LOTOS [17]) for validation.

The paper is organised as follows. Section 2 briefly introduces ET-LOTOS. Section 3 gives a specification model for timed components. Sections 4 and 5 discuss delays and timing constraints respectively. Section 6 explains the characteristics of components specified in different styles using Timed DILL. Section 7 shows how to use Timed DILL to specify and analyse digital logic using a multiplexer as example. Finally, section 8 reviews the work and indicates some future plans.

2 ET-LOTOS in Brief

This section briefly presents ET-LOTOS –a timed LOTOS that allows the modelling of time-sensitive behaviour. A good tutorial introduction is found in [14], while a more theoretical basis is presented in [13].

2.1 Time Specification

ET-LOTOS supports both discrete and dense time domains. Informally, a discrete domain means that time progresses in discrete steps. In a dense domain it is always possible to find a time value between any two given time values. The discrete time domain is represented by the natural numbers, and the dense time domain by the real or rational numbers. However, for ET-LOTOS, only countable time domains (such as rational numbers) are permitted in order to give operational semantics using Labelled Transition Systems.

Three new operators relevant to time are introduced in ET-LOTOS: delay, life reducer and time measurement.

Delay: The delay operator *Delta(time)* means that the subsequent behaviour will be delayed by *time*. In ET-LOTOS a time value is relative to the instant when the previous action occurs. The behaviour *a; Delta(P)* will delay for *d* after event *a* occurs and then behave like *P*.

Time Measurement: The time measurement operator *@t* is used to measure the time elapsed between the instant when the event has been offered and the instant when it occurs. The time value is stored in *t*. In ET-LOTOS, time measurement can be used for both observable actions and internal actions. For observable actions, the time measurement variable *t* can appear in selection predicates. For example *a @t [t $\le$ 5]; P* denotes a process which can perform *a* only within the first 5 time units and then behave like *P*. Otherwise, it will behave like **stop**.

Life Reducer: Applying the life reducer to the internal event, **i** $\{d\}$, means that **i** *must* occur non-deterministically within the next *d* time units. Necessity and non-determinism apply because internal actions are not controlled by the environment; in particular, the time of occurrence is decided by the system. Nonetheless, an alternative action may pre-empt the occurrence of an internal action. If the life reducer is omitted, it is regarded as equivalent to **i** $\{0\}$. As in standard LOTOS, selection predicates cannot be used with the internal action **i**. However, ET-LOTOS introduces the life reducer for internal actions. For behaviour **i** *@t* $\{d\}$*; P* the system must perform **i** within the next *d* time units and then behave like *P*. The time of the internal event will be stored in time variable *t*.

The above operators are primitive to ET-LOTOS. Besides these, ET-LOTOS also provides some shorthand notations for flexibility and convenience. Basically, three kinds of shorthands are offered:

Life Reducer on Observable Actions: The life reducer may be used with an observable action, such as *g* $\{d\}$*; Q*. This is a shorthand for *g @t [t* $\le$ *d]; Q* provided *t* does not appear in *Q*. In this example, *g may* happen within *d* time units. If so the process evolves to *Q*, otherwise the process performs like **stop**. If life reducer is omitted, this means that *g* may occur at any time.

Generalized Life Reducer on Observable Action: *g @t [d1* $\le$ *t* $\le$ *d2]; P* can be rewritten as *g* $\{d1,d2\}$*; P* provided that *t* does not appear in process *P*. Actually, the generalized life reducer can also be expressed as: *Delta(d1) g d2-d1; P*.

Generalized Life Reducer on Internal Actions: The behaviour *Delta(d1)* **i** *@t* $\{d2\}$*; P* can be rewritten as **i** $\{d1,d1+d2\}$*; [t-d1/t] P*, where *t-d1/t* means every *t* appearing in process *P* is replaced by *t-d1*.

2.2 Semantics of ET-LOTOS

The formal semantics of ET-LOTOS is given by a labelled transition system. There are two kinds of transitions: discrete and timed. A discrete transition corresponds to the execution of an action. $P \xrightarrow{a} P'$ means that *P* may perform action *a* and then behave like P'. Timed transitions correspond to the passage of time. If *d* is a variable of sort *Time*, then $P \xrightarrow{d} P'$ means that *P* may idle for *d* then behave like P'. The semantics will not be discussed in detail here since it is found in [13].

ET-LOTOS adopts maximal progress [22] for hidden actions. This has some special importance for Timed DILL. Maximal progress means that if a hidden action can occur, it must happen now (unless an alternative action occurs) and should not be postponed. In other words, hidden actions are urgent in ET-LOTOS. In the DILL approach, each digital component is modelled as a process which is usually connected to others. Input or output ports are modelled by LOTOS gates. Ports used inside a design are hidden and their events become urgent under the assumption of maximal progress.

For other **i** events in ET-LOTOS, urgency is not alway available. In the behaviour **i** $\{d\}$*;* **stop** the internal action can be postponed until *d* time units. But after that, it must happen (unless an alternative action occurs). An internal event is thus urgent only at its upper time bound.

3 Specification Model for Timed Digital Components

Before developing an abstract model to specify timed digital components, it is necessary to consider which timing characteristics need to be specified for a digital design. It is possible to think of characteristics as timing relationships among inputs, among outputs, and among inputs and outputs. The timing relationship from input to output is normally called *delay*. It is the time interval between a signal change on an input and the resulting signal change on an output. A timing relationship among inputs is called a *timing constraint*, meaning that the digital circuit can work correctly only if the constraints are met. In Timed DILL there is no need to specify the timing relationships among outputs directly, as they are determined by delays and timing constraints.

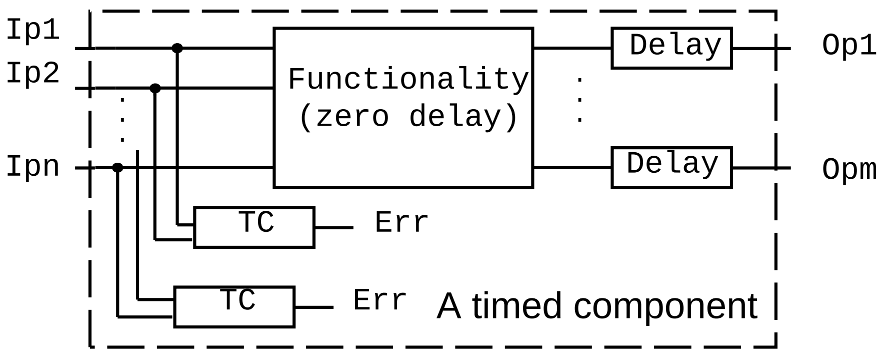

Figure 1: The Specification Model for a Timed Component

There are several possible approaches to specify a timed digital component, classified as either an *integrated method* or a *combined method*. In an integrated method a digital component is specified in one process that deals with both functionality and timing. Although the integrated method may result in compact specifications, it is not a 'structural' method and is hard to apply. The approach is not compositional in the sense that functional and temporal characteristics of a component are interwoven. It is also important to have untimed behaviour as a simple case of timed behaviour, i.e. to be able to isolate pure functionality.

Attention has therefore been focused on developing combined methods. The idea is to separate the functionality and the timing characteristics into different processes, and then combine them in an appropriate way. This makes it possible to reuse the component library developed for Untimed DILL [11] since this mainly deals with functionality. More importantly, the timing work to be discussed in the next two sections can be applied to existing Untimed DILL specifications. Circuit designers can concentrate on specifying the functionality of a logic design, and then incorporate timing characteristics with appropriate parameters.

The initial intention was to use the constraint-oriented style [18, 21] to combine functional and temporal aspects. The functionality part would take advantage of the untimed component, with the timing constraints and delays in parallel. These would constrain whether, when and what outputs should occur, corresponding to timing constraints, delays and functionality respectively. Placing the delay components in parallel with the functionality is unfortunately unworkable in the authors' experience. This is mainly because an event in DILL models *changes* in signal levels but not the levels themselves. Delays cannot simply hold up input signals as it has to be determined whether they cause changes in outputs. This cannot be achieved without knowing the functionality of the component.

The model adopted for Timed DILL was arrived at after considerable experimentation with different approaches. The selected approach is called the *parallel-serial* model. As shown in figure 1, the functionality is assumed to be specified with zero delay. Timing constraints (TC) are placed in parallel with the functional specification to check if inputs requirements are met. Delays are placed in series with the functionality to provide delay for each output.

Note that the *Err(or)* gates in the figure are for analysis purposes only; they have no counterpart in a real physical component. If an *Err* action is offered, it indicates that a timing constraint has not been met. However, the behaviour of the component is not influenced by errors since the functionality part always assumes that inputs meet the constraints. The outputs of the component are thus always 'correct' in terms of the component's function.

This modelling decision was made after careful consideration. The idea is that instead of trying to specify behaviour under all kinds of input conditions, behaviour is specified only for correct inputs. Behaviour under error conditions is 'wrong' and thus not very meaningful. Violation of timing constraints means that there is a design error. Real hardware will do something under these conditions, but the result is not really interesting or relevant. It is more important to detect and correct design errors, not model the behaviour of components under erroneous conditions. The aim of analysis is to find out whether the design is correct, particularly in the presence of timing conditions. During a simulation, the occurrence of an *Err* offer is immediately obvious. For verification, the absence of *Err* offers can be checked before verifying other properties.

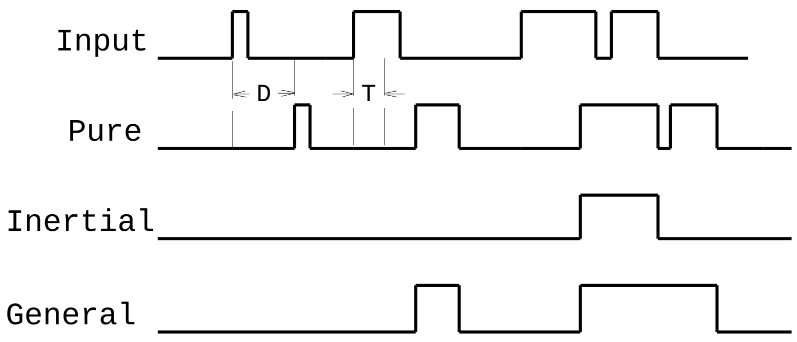

Figure 2: Basic Delay Types

Finally, note that if the timing constraints are void and the delays are between zero and arbitrarily large then the timed model is equivalent to the untimed model. An untimed specification is thus just a special case.

4 Delays

4.1 Basic Delay Types

Because the physical structures of digital components differ, their delays are of different types and values. According to [20] there are two basic delay types: *pure delay* and *inertial delay*. Suppose the delay of a digital component is *D*. If a component has pure delay, all input changes will have an effect on output. In other words, outputs follows inputs after delay *D*. If the component has inertial delay, output will respond only to input changes which have persisted for time *D*. In other words, input pulses whose width is less than *D* will be absorbed by the component. This reflects the fact that short pulses contain insufficient energy to trigger a state change in a real component.

Sometimes, the delay of a component has a more general form. There may exist a threshold $T < D$ such that the component absorbs input pulses whose width is less than *T*. However output follows input if the pulse width is more than *T*. In DILL this is termed *general delay*. In fact, it could be considered as inertial delay *T* cascaded with a pure delay *D-T*. Figure 2 shows how inputs are related to outputs for different delay types. The scale of the figure takes *T* as 2 and *D* as 4 time units. Each input transition has been numbered. As shown in the figure, the pure delay component copies input to output after some interval. The inertial delay component however retains only pulse 5–6 because pulses 1–2, 3–4, 6–7 and 7–8 are all less than *D* time units. For the general delay component, pulses 1–2 are 6–7 are absorbed since they are less than *T* time units.

4.2 Delay Elements in DILL

This section introduces the delay elements that have been included in the DILL library. Although these are components in the sense of building blocks, they are not components like most of the library (gates, flip-flops, counters, etc.). They should perhaps be called pseudo-components. Unlike the fixed delays discussed in the last section, all delays have a non-deterministic range from *MinDel* (the minimum delay) to *MaxDel* (the maximum delay). For general delay, *MinWidth* corresponds to the threshold *T* in the last section. It is obvious that the assumption of non-deterministic delays is more realistic and flexible than that of fixed delays.

4.2.1 Inertial Delay

The following is a naive attempt at specifying a delay. The example reveals some interesting properties of ET-LOTOS which have to be taken into account. The specification uses the ET-LOTOS generalised life reducer to model inertial delay. If the interval between two input transitions is less than the delay, output will not occur. The exact delay will be determined by the environment because the delay range is associated with an observable action. But in DILL what should really be specified is that the delay is decided by the component itself. If the delay is connected to other components in a larger design, the *Op* port might well be hidden. This would mean that the delay time is exactly *MinDel* instead of being a non-deterministic value since ET-LOTOS adopts maximal progress for hidden events.

```
process DelayNaive [Ip, Op] (MinDel, MaxDel : Time) : noexit :=
  DelayNaiveAux [Ip, Op] (MinDel, MaxDel, 0 of Bit, 0 of Bit)
where
  process DelayNaiveAux [Ip, Op]
   (MinDel, MaxDel : Time, DataIp, DataOp : Bit) : noexit :=
    Ip ? NewDataIp : Bit;
    DelayNaive[Ip, Op](MinDel, MaxDel, NewDataIp, DataOp)
  []
    [DataIp ne DataOp] ->
      Op ! DataIp {MinDel, MaxDel};
      DelayNaiveAux [Ip, Op] (MinDel, MaxDel, DataIp, DataIp)
  endproc (* DelayNaiveAux *)
endproc (* DelayNaive *)
```

The following is a better specification of inertial delay:

```
process DelayInertial [Ip, Op] (MinDel, MaxDel: Time) : noexit :=
  DelayInertialAux [Ip, Op] (MinDel, MaxDel, 0 of Bit, 0 of Bit)
where
  process DelayInertialAux [Ip, Op]
   (MinDel, MaxDel : Time, DataIp, DataOp : Bit) : noexit :=
    Ip ? NewDataIp : Bit;
    DelayInertial_Aux [Ip, Op] (MinDel, MaxDel, NewDataIp, DataOp)
  []
    [DataIp ne DataOp] ->
      i {MinDel, MaxDel}; Op ! DataIp {0};
      DelayInertial_Aux [Ip, Op] (MinDel, MaxDel, DataIp, DataIp)
endproc (* DelayInertial *)
```

The specification takes advantage of internal events. The internal event **i** introduces non-deterministic delay, which means the output port can change its value at any time between *MinDel* and *MaxDel*. The exact delay value is determined by the component itself and not by the environment. Moreover even if the component is connected to other components, the delay is still non-deterministic since only hidden events are urgent.

4.2.2 Pure Delay

A pure delay is specified as follows. After an input transition, the pure delay component commits itself to output the transition after an appropriate delay; at the same time it prepares to deal with the next input transition. The interleaving operator ensures that each input transition will have a corresponding output transition no matter if the next input transition appears before the output transition.

```
process DelayPure[Ip, Op] (MinDel, MaxDel : Time) : noexit :=
  DelayPureAux [Ip, Op] (MinDel, MaxDel, 0 of Bit, 0 of Bit)
where
  process DelayPureAux [Ip, Op]
   (MinDel, MaxDel : Time, DataIp, DataOp : Bit) : noexit :=
    Ip ? NewDataIp : Bit;
    ( [NewDataIp eq DataOp] ->
        DelayPureAux [Ip, Op] (MinDel, MaxDel, NewDataIp, DataOp)
    []
```

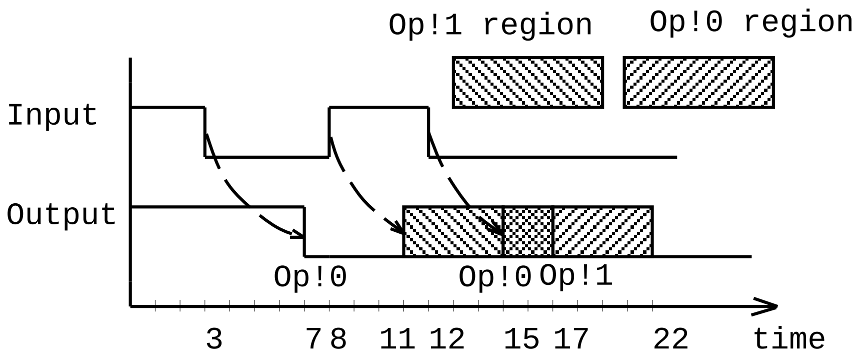

Figure 3: Catch-Up Phenomenon with Pure Delay

```
            [NewDataIp ne DataOp] ->
              ( i {MinDel, MaxDel}; Op ! NewDataIp {0}; stop
              |||
                DelayPureAux [Ip, Op] (MinDel, MaxDel, NewDataIp, NewDataIp)
              ))
      endproc (* DelayPureAux *)
    endproc (* DelayPure *)
```

Because delay is assumed to be non-deterministic rather than fixed, the pure delay above may output ‘strange’ sequences like *Op ! 0; Op ! 0; Op ! 1; . . .* given the input sequence *Ip ! 0; Ip ! 1; Ip ! 0; . . . .* In the sequence, the second *Op ! 0* overtakes *Op ! 1* and results in the two consecutive *Op ! 0* events. The phenomenon of *catch-up* arises if a later input change takes less time to reach the output than an earlier input change. Figure 3 illustrates the reason for the phenomenon, it assumes that the delay is between 3 and 9 time units. As one can see, if both events *Op ! 0* and *Op ! 1* happen within the overlapped region then catch up may arise. Suppose the width of a input pulse is *W*. A necessary condition for catch-up to occur is $W \leq MaxDel\text{-}MinDel$.

Catch-up may exhibit various forms in real hardware if delays vary significantly. However, digital components generally operate in a stable environment so the variation in delays is in a narrow range. Thus the catch-up condition is rarely met in practice. The phenomenon exists in any delay model that is based on pure delay, including the general delay component in the next section. But it does not appear in the inertial delay model since an input change will prevent any pending output; it is therefore not possible to catch up a pending output.

4.2.3 General Delay

As mentioned before, general delay has a threshold *MinWidth*. Input pulses whose width is less than *MinWidth* will be absorbed by the component. They will appear at the output if their width is greater than or equal to *MinWidth*. The general delay element in DILL is specified such that it can model not only a general delay but also inertial or pure delay. This is achieved by choosing appropriate timing parameters. The following specifies the delay component.

```
Process Delay [Ip,Op] (MinWidth, MinDel , MaxDel : Time) : noexit :=
  DelayAux [Ip,Op] (MinWidth, MinDel, MaxDel, 0 of Bit, 0 of Bit)

  where

  process DelayAux [Ip,Op]
   (MinWidth, MinDel, MaxDel : Time, DataIp, DataOp : Bit) : noexit :=
    Ip ? NewDataIp : Bit;
    DelayAux[Ip,Op] (MinWidth,MinDel,MaxDel,NewDataIp,DataOp)
  []
```

```
                [(DataIp ne DataOp)] ->
                  (
                    [MinWidth lt MinDel] ->                    (* general delay *)
                      (
                        Delta(MinWidth) i;                     (* input holds at least MinWidth *)
                        (
                          i {MinDel - MinWidth, MaxDel - MinWidth};
                          Op ! DataIp {0};
                          Stop
                        |||
                          DelayAux [Ip,Op] (MinWidth, MinDel, MaxDel, DataIp, DataIp)
                        )
                      )
                  []
                    [MinWidth ge MinDel] ->                    (* inertial delay *)
                      (
                        i {MinDel, MaxDel} ;
                        Op ! DataIp {0};
                        DelayAux [Ip,Op] (MinWidth, MinDel, MaxDel, DataIp, DataIp)
                      )
                  )
              endproc (* DelayAux *)
            endproc (* Delay *)
```

The rules for timing parameter values can be obtained from the above specification. *Inf* in the following is the maximal value of the time domain (taken as arbitrarily large):

$0 < MinWidth < MinDel \leq MaxDel < Inf$ This describes general delay. Only when *MinWidth* is a positive number less than *MinDel* is the general delay model meaningful.

$MinWidth = 0, MinDel \leq MaxDel < Inf$ This is pure delay. The difference between general delay and pure delay is that for the latter, *MinWidth* is zero so the component does not absorb a narrow pulse.

$0 \leq MinDel \leq MaxDel < Inf, MinWidth > MinDel$ This is inertial delay. It applies if the threshold *MinWidth* is greater than *MinDel*. *MinWidth* is often set to *Inf* for inertial delay(This does not mean that signals are absorbed totally, but just an implementation trick.).

$MinDel = 0, MaxDel = Inf, MinWidth > 0$ This is equivalent to the untimed delay component in the previous version of DILL. Usually *MinWidth* is given the value *Inf*.

4.2.4 High-to-Low and Low-to-High Delays

All the delay models described so far assume the same delay value for transitions from high-to-low (▯) or low-to-high (▯) outputs. Actually, most digital components have different values for them. (This arises because the direction of current flow depends on the transition, and the physical processes have different delays.) If the difference between these delays influences the correctness or performance of a digital circuit, they should be distinguished. In Timed DILL there is a delay element that deals with different ▯ or ▯ delays.

If a component exhibits pure delay, the catch-up phenomenon can still occur. Even if the delays are fixed, the difference between the high-to-low and low-to-high values could result in the faster one catching up the slower one. However, the specification of the delay avoids catch-up for fixed delays. Fixed delays reflect an environment that is stable, so catch-up rarely happens in practice. It is assumed that if a later but faster change catches up with an earlier but slower change, both changes are absorbed by the component. The specification of the delay will not be presented in detail for reasons of space.

Tri-state components are used particularly with buses. They have a high-impedance state when they are not enabled so that several tri-states outputs can be connected. Tri-state components can exhibit different delays when entering or leaving the high-impedance state (Z). As a modelling decision there is no high impedance state in DILL (see [11]), Timed DILL does not have to allow for such delays.

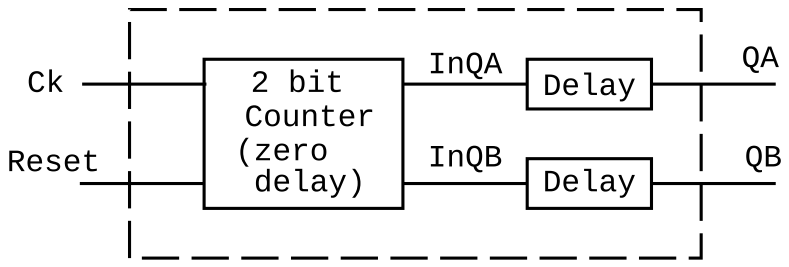

Figure 4: 2-Bit Counter with Asynchronous Reset

4.2.5 Dynamic Delay

Dynamic delay is used when the delay values from several inputs to the same output are different. Typically this happens in a high-level specification since the internal data paths could well be different. Consider a simple 2-bit counter with asynchronous reset. Its two outputs *QA* and *QB* represent a two-bit binary number. If *Reset* is not being requested, the binary output will be incremented after each active clock transition *Ck*. It is set to *00* after a *Reset* request, irrespective of the clock. The timed version of the counter is shown in figure 4. Suppose that a reset is not being requested. When the clock has a positive-going transition, the output will be incremented after some delay (say, 20–30 ns). If a reset is requested the output will change to *00* asynchronously, the delay from clock to output always exceeds the delay from reset to output (say, 10–15 ns).

When a change occurs at the input of any kind of delay (for example *InQA*), the delay component does not know the source of the change (the clock transition or the reset). Therefore it does not know which delay value should be applied (20–30 ns or 10–15 ns). The Timed DILL solves this problem by providing the dynamic delay component specific for such cases. The delay component requires the functional specification to give the delay range when it offers an output change. For example, when the counter is reset, the functionality part must offer *InQA ! 0 ! 10 ! 15*, and the dynamic delay component extracts the delay value from the offer to ensure the appropriate delay occurs.

As one might expect, because different delay values are applied to one delay component the catch-up phenomenon will arise if the component is based on pure delay. The dynamic delay component in DILL library is based only on inertial delay.

5 Timing Constraint Components

Timing constraints in DILL are used to check if the inputs of a component satisfy some conditions. Common timing constraint 'components' have been added to the DILL library, including those for setup, hold, pulse width and period.

Setup and hold times are always associated with flip-flops. For a D (delay) flip-flop, setup time is the time interval between a change in input *D* and the trigger that stores this data (e.g. a positive-going transition of the clock *Ck*). The data signal must then remain stable for a minimal time interval if correct operation of the flip-flop is to be guaranteed. For a flip-flop, the hold time is the interval in which input data must remain unchanged after triggering by the clock. Again, this minimum must be respected for correct operation. A timing diagram showing setup time and hold time is given in figure 5.

The setup time constraint is specified as follows. The hold time constraint is specified in a very similar way. The specification supposes that the active clock transition is positive-going. The width timing constraint defines the minimum width that an input pulse can have. A call of the process *Width [Ip, Err] (MinWidth, 1)* checks that the input *Ip* stays at 1 for at least *MinWidth* in each cycle. The period timing constraint is the minimum period for an input signal, and is always used for a clock signal. For process *Period [Ck, Err] (MinPeriod)* the minimum interval between two consecutive positive-going (or negative-going) transitions should be at least *MinPeriod*. Figure 6 illustrates the two constraints.

```
process SetupDel [D, Ck, Err] (SetupTime : Time) : noexit :=
  D ? NewDataIp: Bit;                              (* new data input *)
```

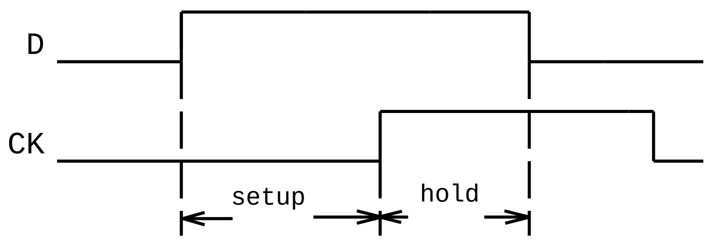

Figure 5: Setup and Hold Times for D Flip-Flop

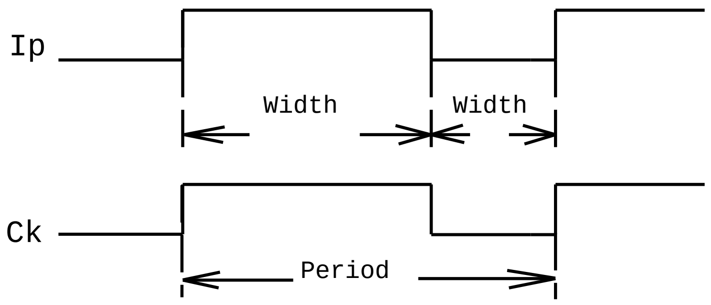

Figure 6: Width and Period timing constraints

```
  AfterD [D, Ck, Err] (SetupTime, SetupTime)          (* check setup time *)
[]
  Ck ? NewCk : Bit;                                   (* new clock input *)
  SetupDel [D, Ck, Err] (SetupTime)                   (* no setup time to check *)
endproc (* SetupDel *)

process AfterD [D, Ck, Err] (SetupTime, SetupRem : Time) : noexit :=
  Delta(SetupRem) i;                                  (* enforce min. setup time *)
  SetupDel [D, Ck, Err] (SetupTime)                   (* restart setup check *)
[]
  Ck ? NewCk : Bit @ t;                               (* new clock input *)
  ([NewCk eq 0] ->                                    (* negative-going clock? *)
   AfterD [D, Ck] (SetupTime, SetupTime - t)          (* check remaining setup time *)
  []
   [NewCk eq 1] ->                                    (* positive-going clock *)
    Err ! SetupError;                                 (* min. setup time violated *)
    SetupDel [D, Ck, Err] (SetupTime))                (* restart setup check *)
[]
  D ? NewDataIp: Bit;                                 (* new data input *)
  AfterD [D, Ck, Err] (SetupTime, SetupTime)          (* restart setup check *)
endproc (* AfterD *)
```

6 The Timed DILL Library

The new delay components can be used along with the untimed components in the DILL library. None of the library components give timing constraints, so the user must introduce these to check input timing.

Section 6.2 and 6.3 are related to components in different specification styles. The structural style specifies a component as the connection of lower-level components. It thus indicates how a digital circuit is constructed. The data flow style describes how input signals flow through the circuit to produce the outputs. Normally, each output can be specified by an expression using the LOTOS **let . . . in** operator. Finally, the behavioural style defines components through their input-output response. It specifies a black-box digital component whose inputs and outputs interact with the environment. More information on these styles can be found in [11].

6.1 Basic Logic Gates

For Timed DILL, the basic logic gates (including tri-state ones) are built from a zero-delay functionality part and a general delay part. The zero-delay functionality is easily obtained from the component in the Untimed DILL: a life reducer {0} is appended to each output event offer. Because the basic logic gates are very common in logic design, general delay was chosen so that timed components could be used in different designs with different delay assumptions. The actual timing characteristics are obtained by defining the timing parameters mentioned in section 4.2.3.

Most actual designs use basic logic gates from the same technology family, e.g. CMOS (Complementary Metal-Oxide Silicon). This means that timing parameters are defined by the choice of technology. It would be clumsy to have to specify timing parameters for each logic gate individually. The technology is therefore specified at a global level in DILL. Timing parameters are then supplied automatically for basic logic gates. Even for components which are specified abstractly, when their specification is expanded to the gate level by DILL, the timing parameters are added by default. However, DILL specifiers can also specify timing characteristics directly if required.

6.2 Components Specified in Structural Style

Components in the structural style are defined by combining lower-level components. This determines timing characteristics are derived from the timing characteristics at a lower level. For this reason, timed components in the structural style do have not timing parameters associated with them. This does not, however, mean that the component is untimed: its timing properties are determined at the gate level.

6.3 Components Specified in Abstract Style

Data-flow style components have their outputs described by an expression. Suppose that an output value is calculated. The style does not uniquely distinguish which input (or output) results in the transition. This means it is not possible to state a precise pin-to-pin delay, the delay from a specific input to a specific output. The delay component associated with each output port has to use a common delay range, i.e. all paths from the inputs to that output have to be treated as having the same delay value.

The pin-to-pin delay can, however, be defined for behaviourally specified components. In order to specify such delay, the functional specification needs some changes. After each input offer that may lead to output, each output is checked to see if its value has been influenced by the input transition. If so, there should be an event offer with the specific delay from *this* input to *this* output. As for the counter example in section 4.2.5, dynamic delays should be used in this case.

7 Timed DILL Example: A 2-to-1 Multiplexer

7.1 2-to-1 Multiplexer

As an example, the 2-to-1 multiplexer in the DILL library will be specified and analysed. This component has two data inputs *A* and *B*, a selection input *S* and an output *C*. The behaviour is such that if the selection input is 0, the data at *A* will appear at *C* after some delay. Alternatively if the selection input is 1, the data at B will appear at *C*. The delays used in the example are inertial, mainly because they are easy to handle but are general enough to represent delay in most digital circuits.

The multiplexer can be specified at two levels. The higher level specifies the required behaviour and timing performance. The lower level specifies the structure of the component by connecting basic logic gates. The lower level implements the higher level. The timed specifications were analysed through simulation and testing.

7.2 Timed LOTOS Tools

This section shows how Timed DILL can be used to specify and analyse digital designs that are time-sensitive. The objective is to discover and correct timing hazards. Alternatively the nature of the hazards can be used to determine the settling time required before the outputs of a design are stable.

In principle, all the examples developed for Untimed DILL [11] can be reworked for Timed DILL. Unfortunately, tools supporting ET-LOTOS are currently under development. The tool available to the authors was TE-LOLA (Time Extended LOTOS Laboratory [16]), which supports TE-LOTOS (Time Extended LOTOS [17]). However, Christian Hernalsteen (University of Brussels) also kindly investigated the feasibility of analysing Timed DILL specifications using the model and tool he has under development [5]. This looks like a promising approach in future.

It has been possible to use TE-LOLA to analyse the specifications generated by Timed DILL. Although ET-LOTOS and TE-LOTOS adopt different semantic models, the equivalence between them has been established [15]. It is therefore possible to translate the generated ET-LOTOS specifications into TE-LOTOS syntax. Because of their similarity, the translation is always possible although some subtle differences need attention. For example, **i** $\{d\}$ in ET-LOTOS means **i** will happen non-deterministically between *0* and *d* time units, but in TE-LOTOS it means that **i** will occur at exactly time *d*. The correct translation from ET-LOTOS should be to **i** $\{0..d\}$ in TE-LOTOS. In order to avoid confusion, the following specifications will use ET-LOTOS though the actual analysis was made with TE-LOTOS.

7.3 Behavioural Specification

7.3.1 DILL Description

The behavioural specification of the 2-to-1 multiplexer in DILL is as follows. DILL provides a veneer on top of LOTOS – mainly a library of components that can be combined using LOTOS operators. The **circuit** declaration names the overall specification and its parameters. It then gives a LOTOS behaviour expression for the whole circuit. Library components are automatically included by giving their names (*Component_Decl*). Here, *Multiplexer2to1_BB_0* is a 2-to-1 multiplexer in black-box form that exhibits zero delay. It was derived from the corresponding component in Untimed DILL using the approach in section 6.1. The behavioural specification defines an inertial delay between *MinDel* (10) and *MaxDel* (15).

```
define(MinDel, 10)                                  # min. delay value
define(MaxDel, 15)                                  # max. delay value
include(dill.m4)                                    # include DILL library
circuit(                                            # circuit description
  ‘Multiplexer2to1_BB [A, B, S, C]’,‘               # circuit name and ports
  hide InC in                                       # internal gate to delay
    Multiplexer2to1_BB_0 [A, B, S, InC]             # multiplexer instance
  |[InC]|                                           # sync with delay
    Delay [InC, C] (Inf, MinDel, MaxDel)            # delay instance
  where
    Multiplexer2to1_BB_0_Decl                       # multiplexer from library
’)
```

7.3.2 Validation

The behavioural specification was validated using the TE-LOLA step-by-step simulator. Basically, the behaviour of the multiplexer is simulated for each input combination to see if it is as expected. The results of simulation are regarded as the criteria against which simulation of the lower-level specification should be judged. As an example, the trace after simulating the behaviour corresponding to input state *A=1, B=1, S=0* is as follows:

```
[1] - A ! 1 {0};
[2] - B ! 1 {0};
[3] - S ! 0 {0};
```

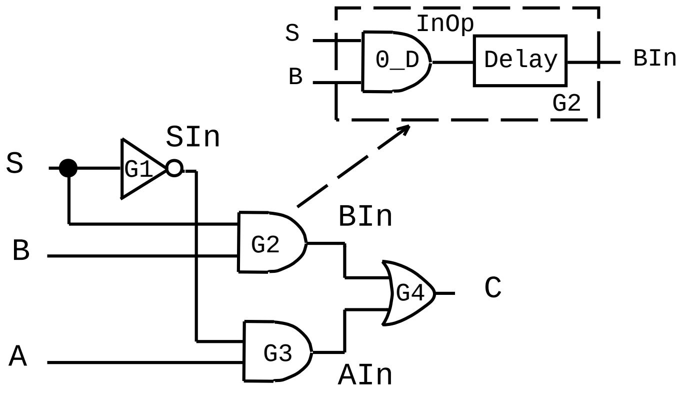

Figure 7: Structure of 2-to-1 Multiplexer as Timed Logic Gates

```
[4] - i {0};                          (* InC ! 1 *)
[4] - i @ t [10 <= t <= 15];
[4] - C ! 1 {0};
```

Here, internal event *InC* is the input to the delay component. This trace indicates that for the input state *110*, the output *C* should be *1* after a delay between 10 to 15 time units. Other input states can be simulated in the same way.

Because test results from TE-LOLA become inconclusive if tests have actions with intervals such as **i** {*5,7*}, it is possible to use only fixed delays during testing. For a higher-level specification like the behavioural one, such an assumption would be unrealistic. There are several paths from the inputs to an output, so the delay associated with the output has a range of values.

7.4 Structural Specification

7.4.1 DILL Description

The structure of the 2-to-1 multiplexer is shown in figure 7.[1] The logic gates in the diagram are timed gates. Each of them consists of zero-delay logic and a delay component. The inset in the figure shows the structure of the *and* gate *G2*; *0_D* in the figure means zero delay. Other gates have the same kind of structure. The design of the multiplexer is 'classical' and can be found in textbooks like [12]. However, as will be seen later this design contains timing hazards.

Assuming the delay of each gate is 5 time units, the corresponding DILL specification is as follows. The specification first defines the delay of the basic logic gates. Here the delay is fixed at 5 and is inertial because *MinWidth* is *Inf*. The first parameter of the **circuit** declaration is optional. In this example it is **timed**, indicating that basic logic gates use the delay data defined by the specifier. Other possible values of the parameter are **CMOS**, **ECL** and **TTL**. Delay data for these logic families is pre-defined in Timed DILL. The default value for a circuit is **untimed**, which appends *(Inf, 0, Inf)* to every instantiation of a basic logic gate. For a higher-level specification like the one in section 7.3 there is no need to define the delay parameters because it does not use basic logic gates. But this does not means that the component is untimed.

```
define(DelayData, '(Inf, 5 of Time, 5 of Time)')     # delay values
include(dill.m4)                                      # include DILL library
circuit(                                              # circuit description
  timed,                                              # declare timed design
  'Multiplexer2to1 [A, B, S, C]','                    # circuit name and ports
  hide AIn, BIn, SIn in                               # internal gates
    Inverter [S, SIn]                                 # inverter instance
```

[1]The triangle with a circle is an inverter, the 'D' shapes are *and* gates, the shield shape is an *or* gate.

```
    |[S, SIn]|                                    # sync with selection signals
      (
        And2 [SIn, A, AIn]                          # two-input and instance
      |||
        And2 [S, B, BIn]                            # two-input and instance
      )
    |[AIn, BIn]|                                  # sync with inputs
      Or2 [AIn, BIn, C]                             # two-input or instance
    where
      Inverter_Decl                                 # inverter from library
      And2_Decl                                     # two-input and from library
      Or2_Decl                                      # two-input or from library
’)
```

7.4.2 Validation

Timed behaviour was investigated using the *TextExpand* function of TE-LOLA. Testing was done by composing test processes in parallel with the specification. If the test process can be followed for all executions of the composed specification, the result of testing is *must pass*. If the test process can be followed only for some executions, the result is *may pass*. Otherwise the test is considered to be *rejected*.

First, the functionality of the multiplexer was tested. Eight test processes are defined to check if the output correctly corresponds to each possible input combination. For example, for *A=1, B=1, S=0* the output should be *C=1* after 10 or 15 time units (because the input-output paths contain 2 or 3 levels of basic gate). The following test process corresponds to the the *110* state.

```
process Test110_1 [A, B, S, C, OK] : noexit :=
  A ! 1 {0}; B ! 1 {0}; S ! 0 {0};
  (C ! 1 @ t [t = 10]; OK; stop
  []
   C ! 1 @ t [t = 15]; OK; stop)
endproc (* Test110_1 *)
```

The *OK* event is used to denote success in the *TextExpand* function of TE-LOTOS; it is not an input or output of the circuit. The test successfully passed as expected. The other 7 input combinations were tested in a similar way.

Second, there were tests to see if the design had a timing hazard. Hazards are unwanted transitions that appear on the outputs of digital circuits in response to the changes on inputs. For example, suppose that the output should stay the same (e.g. *1*) after the input state changes from I_1 to I_2. However, in an actual implementation the output may change from *1* to *0* and back again after an input transition. The consecutive unwanted transitions *1* to *0* and *0* to *1* are hazards. The multiplexer has 3 input ports and thus 8 input states. Each input state may change to one of the other 7 input states, so there are 56 possible input transitions (8 $\times$ 7) in total. For each transition, a test process is used that *risks* hazards. If the design of the multiplexer is hazard-free, then each test process should be rejected. Evaluating the 56 test processes showed that 6 of them pass the test, i.e the circuit is not hazard-free. Table 1 lists these transitions and the corresponding hazards. The test results indicate that when the delays of each gates are fixed, the circuit exhibits static hazards. One of the hazards happens when there is a single input change; the others occur when more than one input changes simultaneously.

The test corresponding to the transition 111 to 110 (*A=1, B=1, S=1* to *A=1, B=1, S=0*) looks like:

```
process Test111_110Hazard [A, B, S, C, OK] : noexit :=
   (A ! 1 {0}; B ! 1 {0}; S ! 1 {0};
    (C ! 1 @ t [t = 10]; exit
    []
     C ! 1 @ t [t = 15]; exit))                  (* state 111 *)
  >>
   (S ! 0 @ t [t = 2];                           (* input change, state 110 *)
```

Transition	Type of Hazard	Changed Inputs
000 to 101	static-0	2
010 to 101	static-0	3
011 to 100	static-1	3
011 to 110	static-1	2
111 to 100	static-1	2
111 to 110	static-1	1

Table 1: Hazards in the 2-to-1 Multiplexer

```
      (C ! 0 @ t [t = 10];                                (* hazard *)
        C ! 1 @ t [t = 5]; ok; stop
       []
        C ! 0 @ t [t = 15];                               (* hazard *)
        C ! 1 @ t [t = 5]; ok; stop))
  endproc (* Test111_110Hazard *)
```

The output transitions *C ! 0* and *C ! 1* in the process indicate a hazard because the output should remain at 1 for the transition 111 to 110.

By analysing the passed test sequences it is discovered that hazards are caused by inputs following different lengths of path to reach the output. Figure 8 is a hazard-free design of the 2-to-1 multiplexer.[2] Three additional repeaters are used to guarantee that each input-output path has exactly three gate delays. It is obvious that the delay of each repeater should also be 5 time units. The total delay of the new design is 15 time units, which complies with the timing constraint expressed by the behavioural specification.

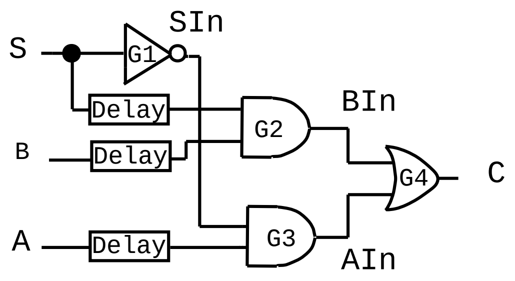

Figure 8: The Hazard-Free Multiplexer

8 Conclusion and Future Work

A timed extension of DILL has been presented. First, the timing characteristics of digital circuits were investigated and categorised as timing constraints and delays. Then the parallel-serial model was used to specify timed components. To make Timed DILL more practical for digital design, several commonly used timing constraints and delay types have been introduced into the DILL library. The characteristics of these pseudo-components have been explained. Finally, a multiplexer example has been used to illustrate the specification and analysis of timed digital logic.

Timed DILL offers a number of important benefits. First, it can be used to check whether timing requirements on a digital design are respected. This can be done using the timing constraint components. Second, it can be used to discover potential timing errors like hazards, as in the multiplexer example. Third, it can be used to analyse the timing properties of a logic design such as its minimal and maximal delays.

[2]The triangle is a repeater whose output always follows its input.

To gain these benefits really needs the help of tools. A future goal is to use verification tools with Timed DILL. One possibility is KRONOS [2], which checks if the system described by a timed automaton satisfies a requirement expressed as a formula of TCTL (Timed Computational Tree Logic [1]). A method for transforming ET-LOTOS specifications to timed automata has already been implemented by others [3, 5]. Verification of a Timed DILL specification may thus be possible. However, as KRONOS is a tool that supports only specifications without state variables, it may be necessary to resort to other verification tools such as Hytech [4] which supports variables. This would need translation of ET-LOTOS into a Hytech automaton – work that is currently being undertaken by others [5].

Acknowledgements

Ji He gratefully acknowledges financial support from the Sino-British Fellowship Trust and the University of Stirling during much of the work reported in this paper. The authors thank Christian Hernalsteen (University of Brussels) and Guy Leduc (University of Liège) for their advice on use of tools with ET-LOTOS specifications. Gualberto Rabay Filho (Polytechnic University of Madrid) kindly gave assistance with TE-LOLA. Carron Shankland (University of Stirling) provided helpful comments on a draft of the report.

References

[1] R. Alur, C. Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Proc. 5th Symposium on Logic Computer Science*, pages 414–425, 1990.

[2] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In B. Mounier, J. L. Albert, and M. Rodriguez, editors, *Proc. Hybrid System III*, volume 1066 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1996.

[3] C. Daws, A. Olivero, and S. Yovine. Verifying ET-LOTOS programs with KRONOS. In Richard L. Tenney, Paul D. Amer, and M. Ümit Uyar, editors, *Proc. Formal Description Techniques VI*, pages 227–242. Chapman-Hall, London, UK, 1995.

[4] T. A. Henzinger, Pei-Psin Ho, and G. Wong-Toi. Hytech: The next generation. In *Proc. 16th Real-Time Systems Symposium*, pages 56–65. Institution of Electrical and Electronic Engineers Press, New York, USA, 1995.

[5] Christian Hernalsteen. A timed automaton model for ET-LOTOS verification. In Tadanori Mizuno, Norio Shiratori, Teruo Higashino, and Atsushi Togashi, editors, *Proc. Formal Description Techniques X/Protocol Specification, Testing and Verification XVII*, pages 193–204. Chapman-Hall, London, UK, November 1997.

[6] IEEE. *VHSIC Hardware Design Language*. IEEE 1076. Institution of Electrical and Electronic Engineers Press, New York, USA, 1992.

[7] IEEE. *IEEE Standard Hardware Design Language based on the Verilog Hardware Description Language*. IEEE 1364. Institution of Electrical and Electronic Engineers Press, New York, USA, 1995. ISBN 1-55937-727-5.

[8] IEEE. *Standard VITAL ASIC Modelling Specification*. IEEE 1076.4. Institution of Electrical and Electronic Engineers Press, New York, USA, 1995.

[9] Open Verilog International. Standard Delay Format specification. Technical Report SDF Version 3.0, Open Verilog International, Suite 109-071, 15466 Los Gatos Boulevard, Los Gatos, CA 95032, USA, May 1995.

[10] ISO/IEC. *Information Processing Systems – Open Systems Interconnection – Enhancements to LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*. ISO/IEC CD. International Organization for Standardization, Geneva, Switzerland, February 1998.

[11] He Ji and Kenneth J. Turner. Extended DILL: Digital logic with LOTOS. Technical Report CSM-142, Department of Computing Science and Mathematics, University of Stirling, UK, November 1997.

[12] Raymond Kline. *Structured Digital Design, Including MSI/LSI Components and Microprocessors*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1983.

[13] Luc Léonard and Guy Leduc. An enhanced version of timed LOTOS and its application to a case study. In Richard L. Tenney, Paul D. Amer, and M. Ümit Uyar, editors, *Proc. Formal Description Techniques VI*, pages 483–500. North-Holland, Amsterdam, Netherlands, 1994.

[14] Luc Léonard and Guy Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29(2):271–292, February 1997.

[15] Luis Llana and Gualberto Rabay Filho. Defining equivalences between time/action graphs and timed action graphs. Technical report, Department of Telematic Systems Engineering, Polytechnic University of Madrid, Spain, December 1995.

[16] Santiago Pavón, David Larrabeiti, and Gualberto Rabay Filho. LOLA user manual (version 3R6). Technical report, Department of Telematic Systems Engineering, Polytechnic University of Madrid, Spain, February 1995.

[17] Gualberto Rabay Filho and Juan Quemada. TE-LOLA: A timed LOLA prototype. In Zmago Brezocnik and Tatyana Kapus, editors, *Proc. COST 247 International Workshop on Applied Formal Methods*, pages 85–95, Slovenia, June 1996. University of Maribor.

[18] Kenneth J. Turner. Incremental requirements specification with LOTOS. *Requirements Engineering Journal*, 2:132–151, November 1997.

[19] Kenneth J. Turner and Richard O. Sinnott. DILL: Specifying digital logic in LOTOS. In Richard L. Tenney, Paul D. Amer, and M. Ümit Uyar, editors, *Proc. Formal Description Techniques VI*, pages 71–86. North-Holland, Amsterdam, Netherlands, 1994.

[20] Stephen H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley-Interscience, UK, 1969.

[21] Chris A. Vissers, Giuseppe Scollo, and Marten van Sinderen. Architecture and specification style in formal descriptions of distributed systems. *Theoretical Computer Science*, 89:179–206, 1991.

[22] Yi Wang. CCS + time = An interleaving model for real-time systems. In *Proc. 18th International Colloquium on Automata Languages and Programming*, number 510 in Lecture Notes in Computer Science, pages 217–228. Springer-Verlag, Berlin, Germany, 1991.

A Delay and Timing Constraints in the Library

This section summarises the pseudo-components for timing that have been placed in the DILL library. Some of the components are redundant since, for example, *Delay_Inertial* and *Delay_Pure* are included in the *Delay*. But using these components is more convenient in some specifications.

In table 2, value parameters *Edge* and *LevelOfPulse* are of sort *Bit*; they indicate the active clock transition and the level of the checked pulse respectively. *1/0* represents a positive-going/negative-going transition or a *1/0* level pulse. The other value parameters are of sort *Time*.

Pseudo-Component	**Meaning**
Delay [Ip, Op] (MinWidth, MinDel, MaxDel)	General delay
Delay_Dyn [Ip, Op] (MinDel, MaxDel)	Dynamic delay
Delay_Inertial [Ip, Op] (MinDel, MaxDel)	Inertial delay
Delay_HL [Ip, Op] (MinWidth, MinT01, MaxT01, MinT10, MaxT10)	⌜and ⌞delay
Delay_HL_Inertial [Ip, Op] (MinT01, MaxT01, MinT10, MaxT10)	Inertial ⌜and ⌞delay
Delay_Pure [Ip, Op] (MinDel, MaxDel)	Pure delay
HoldDel [D, Ck, Err] (HoldTime, Edge)	Hold time constraint
Period [Ck, Err] (MinPeriod)	Period constraint
SetupDel [D, Ck, Err] (SetupTime, Edge)	Setup time constraint
Width [Ip, Err] (MinWidth, LevelOfPulse)	Width constraint

Table 2: Timing Pseudo-Components