From Ideal to Realisable Real-Time Specifications

Graeme Smith

Software Verification Research Centre
University of Queensland, Australia
smith@it.uq.edu.au

Abstract. Formally refining a real-time specification to an implementation is only possible when the specification allows for all physical limitations, and timing and signal errors inherent in the implementation. Allowing for such implementation-specific details in a top-level specification can, however, obscure the desired functionality and complicate analysis. Furthermore, such an approach assumes the specifier has an understanding of the physical limitations and errors of the implementation which may not yet have been developed. As an alternative, we propose introducing a notion of *realisation* into the formal development process. Realisation is an approach to specification development which allows errors and physical limitations to be introduced. It also allows properties of the new specification to be derived from those proved for the original.

1 Introduction

Several formal notations have been developed for the specification and refinement of real-time systems [19, 15, 16, 3, 13, 12]. To simplify specifications and subsequent analysis, most published case studies using these notations ignore the signal and timing errors which are inherent in any implementation, specifying instead ideal, error-free systems.

Such a simplifying approach to modelling and analysing systems is commonly used in traditional engineering disciplines where errors are assumed to be small enough that they have little, if any, effect on the overall behaviour. However, these disciplines do not require the model to be formally refined to an implementation. When this is required, allowance for potential errors must be present in the specification since an ideal specification is not *realisable*, i.e., it cannot be refined to an implementation.

To allow for the errors at the highest level of abstraction, however, obscures the desired functionality of the system and complicates any attempt at analysis. It also requires the specifier to be aware of all aspects of the future implementation which may result in such errors. This is obviously not possible if the specification is being prepared in order to develop the implementation.

In this paper we introduce an alternative approach based on a new notion of *realisation*. Like refinement, the goal of realisation is to transform a specification to one which is closer to an implementation in such a way that properties of

the new specification can be derived from those proved for the original. Unlike refinement, however, the functionality of the original specification is not maintained; instead, an acceptable approximation to this functionality is produced. This allows an ideal specification to be changed to one which can be readily refined to an implementation.

The ideas in this paper were inspired by Hayes's paper on specifying physical limitations [9]. In that paper, a Z [18] specification of an ideal oscilloscope is used in the specification of an oscilloscope with limitations on its frequency and voltage response, and errors in its timing and output. We present a general approach for introducing such physical limitations and errors to specifications written in the Z-based timed refinement calculus [13, 12].

The timed refinement calculus specification notation is reviewed in Section 2. Methods for introducing physical limitations, timing errors and signal (i.e., input and output) errors to specifications are presented in Sections 3, 4 and 5 respectively. Conclusions and related work are discussed in Section 6. We use a simple hardware device as an example throughout the paper. The techniques, however, are equally applicable to real-time software systems.

2 Timed refinement calculus

The basis of the timed refinement calculus is the work of Mahony and Hayes [13, 12]. This has been extended with a simple set-theoretic notation for concisely expressing time intervals by Millerchip et al. [14] and operators for accessing interval endpoints by Duddy et al. [5]. We adopt a simplified subset of the formalism suggested by Fidge et al. [7] which aims at providing a minimal set of operators outside those of standard set theory.

Absolute time $\mathbb{T}$ is modelled by real numbers.

$$\mathbb{T} == \mathbb{R}$$

Observable variables of a system are modelled as total functions from the time domain to a type representing the set of all values the variable may assume. For example, consider the simple sample-and-hold device of Figure 1.

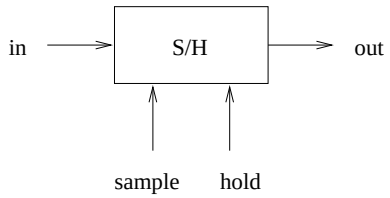

Figure 1: Sample-and-hold device

We assume that sample and hold commands occur for single points of time: in an implementation these point might correspond, for example, to the rising or falling edges of a pulse. When the sample command occurs, the device samples the input signal *in*. This continues until the hold command occurs. At this time

the output signal *out* becomes equal to the currently sampled value of *in* and remains at this value until the next sample command. This is illustrated in Figure 2.

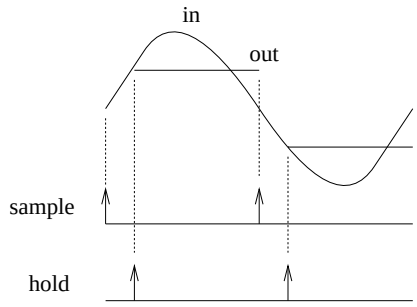

Figure 2: Sample-and-hold behaviour

A variable *sample* representing whether a sample command is being applied to a sample-and-hold device may be declared, using the Boolean type $\mathbb{B}$, as

$$sample : \mathbb{T} \rightarrow \mathbb{B}.$$

A variable *hold* could be similarly declared and, using a type $Volts == \mathbb{R}$, the output to a sample-and-hold device may be declared as

$$out : \mathbb{T} \rightarrow Volts.$$

Functions modelling physical quantities generally map from the time domain to some contiguous subset of the real numbers. In most cases, such functions are *differentiable* (i.e., continuous and smooth). To facilitate specifying this, whenever X is a contiguous set of real numbers we use the notation $\mathbb{T} \rightsquigarrow X$ to represent the set of all differentiable functions from the time domain to X [6]. For example, the input to a sample-and-hold device may be declared as

$$in : \mathbb{T} \rightsquigarrow Volts.$$

A system is specified by constraints on the time intervals over which properties hold. Sets of such intervals can be specified using the interval brackets $(\!|$ and $|\!)$. (The combination of round and square brackets reminds us that the interval end point is allowed to be either open or closed $[14, 7]$.) For example, the set of all intervals where the sample command is applied for the whole interval is specified as

$$(\!| sample |\!)$$

and the set of all intervals where the sample command is applied and the input is greater than 10 volts for the whole interval is specified as

$$(\!| sample \land in > 10 |\!).$$

In general, the property in the brackets is any first-order predicate in which variables declared as total functions from the time domain to some type X may be treated as variables of type X. The elision of explicit references to the time domain of these functions results in specifications which are more concise and readable.

The starting point, end point and duration of intervals can also be accessed using the reserved symbols α, ω and δ respectively. For example, the set of all intervals starting at time 10 and ending at time 15 is specified as

$$[\![\alpha = 10 \land \omega = 15]\!]$$

and the set of all intervals during which the sample command is not applied for a duration of 10 time units is

$$[\![\neg\ sample \land \delta = 10]\!].$$

Predicates are formed by combining sets of intervals using operators from set theory such as $\cap$, $\cup$ and $\subseteq$. For example, the hold property of the sample-and-hold device is specified by the following Z schema.

SampleAndHold

$$sample, hold : \mathbb{T} \to \mathbb{B}$$
$$in : \mathbb{T} \rightsquigarrow Volts$$
$$out : \mathbb{T} \to Volts$$

$$[\![hold(\alpha) \land \neg\ sample]\!] \subseteq [\![out = in(\alpha)]\!]$$

That is, the set of all intervals where hold is applied at the beginning of the interval and sample is not applied is a subset of the set of all intervals where the output is equal to the input at the beginning of the interval for the whole interval.

Specifications in the timed refinement calculus may include assumptions about the environment in which the specified system is to operate. For example, to complete the specification of the sample-and-hold device, we need to place some assumptions on the variables *sample* and *hold*. Ideally, the commands should be applied for a single point of time only and never at the same time. These properties are captured by the following Z schema. (These assumptions are sufficient to illustrate our points. Other assumptions about the ordering and temporal spacing of commands, for example, could also be made.)

Commands

$$sample, hold : \mathbb{T} \to \mathbb{B}$$

$$[\![sample]\!] \cup [\![hold]\!] \subseteq [\![\delta = 0]\!]$$
$$[\![sample \land hold]\!] = \varnothing$$

That is, all intervals in which *sample* or *hold* are true have zero duration and hence comprise a single point of time. Also, there are no intervals where both *sample* and *hold* are true for the whole interval.

The sample-and-hold device is specified as follows.

$$SH \mathrel{\widehat=} out : [Commands, SampleAndHold]$$

Such a specification statement comprises a list of output variables (in this case just *out*) and two predicates. The first predicate is the *assumptions* the specification makes about the environment. It may not refer to the output variables. The second is the *effect* of the specified system under these assumptions.

Given a specification statement $\vec{x} : [A, E]$, all properties of the specified system can be expressed in the form $S \Rightarrow T$ where S describes constraints on the inputs which also satisfy the specification assumption, i.e., $S \Rightarrow A$, and T describes a relationship between inputs and outputs which is implied by the specification effect when S is true, i.e., $S \wedge E \Rightarrow T$. For example, since

$$(\forall t : \mathbb{T} \bullet sample(t) \Leftrightarrow t \bmod 2 = 0 \wedge hold(t) \Leftrightarrow t \bmod 2 = 1) \\ \Rightarrow Commands$$

and

$$(\forall t : \mathbb{T} \bullet sample(t) \Leftrightarrow t \bmod 2 = 0 \wedge hold(t) \Leftrightarrow t \bmod 2 = 1) \\ \wedge SampleAndHold \Rightarrow out(3.5) = in(3)$$

the following is a property of SH.

$$(\forall t : \mathbb{T} \bullet sample(t) \Leftrightarrow t \bmod 2 = 0 \wedge hold(t) \Leftrightarrow t \bmod 2 = 1) \\ \Rightarrow out(3.5) = in(3)$$

The fundamental rules for refinement in the timed refinement calculus are weakening of the assumptions and strengthening of the effects [13]. Hence, any properties of a specification are also properties of its refinements.

Other notation of the timed refinement calculus includes the concatenation operator ';' which operates on intervals, and the piping operator '$\gg$' and hiding operator '$\backslash$' which operate on specification statements.

The concatenation operator ';' joins intervals from two sets, to form the intervals of another set, provided their end points meet. (One end point must be closed and the other open [7].) For example, the set of all intervals in which $x : \mathbb{T} \rightarrow \mathbb{B}$ becomes true and remains true is specified as follows.

$$[\![\neg\, x]\!] ; [\![x]\!]$$

The piping operator '$\gg$' allows the effect of one specification statement to establish the assumption of another. For example, a system S which amplifies a signal by 4 times provided the resulting signal does not exceed 10, can be specified in terms of two specification statements which amplify a signal by 2 times as follows. (Note that the set $[\![true]\!]$ contains all possible intervals.)

$$S \mathrel{\widehat=} y : [[\![u < 2.5]\!] = [\![true]\!], [\![y = 2 * u]\!] = [\![true]\!]] \\ \gg \\ x : [[\![y < 5]\!] = [\![true]\!], [\![x = 2 * y]\!] = [\![true]\!]]$$

The output y of the first specification statement is used as an input to the second. It should be noted, however, that there is no implied temporal ordering between the specification statements, i.e., piping does not correspond to sequential composition. The piping operator can be defined as follows [12, Chapter 4].

$$\vec{x} : [A_1, E_1] \gg \vec{y} : [A_2, E_2] = \vec{x}, \vec{y} : [A_1 \wedge \forall \vec{x} \bullet E_1 \Rightarrow A_2, E_1 \wedge E_2]$$

The hiding operator '$\backslash$' enables one or more output variables to be hidden from the environment of the specified system. Such variables are referred to as *local variables*. For example, the intermediate output y of the above specification S can be hidden as follows. (The brackets $|[\ ]|$ enclose the local environment in which y may be referenced.)

$$|[S \backslash \{y\}\ ]|$$

Hiding may be defined as follows.

$$|[\vec{x}, \vec{y} : [A, E] \backslash \vec{y}\ ]| = \vec{x} : [A, \exists\, \vec{y} \bullet E]$$

3 Physical limitations

The simplest implementation of a sample-and-hold device is illustrated by the circuit in Figure 3 [11, Section 2.4]. A switch is closed by the sample command and opened by the hold command. When closed a capacitor C is charged (through a resistor R) to the input voltage. When the switch is opened, the output is equal to the voltage the capacitor was charged to.

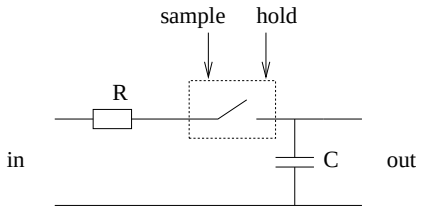

Figure 3: Circuit for sample-and-hold device

The components of this circuit, i.e., the resistor, capacitor and switch (typically implemented by a transistor), would be limited in the voltages they could operate under. Hence, there should be a limit on the maximum voltage of the input signal. Such a limit was not considered in the ideal specification of Section 2 and it would have been inappropriate to include it at that level of abstraction. However, the limit is required in the implementation and cannot be introduced via refinement since adding the restriction on *in* would strengthen the specification's assumption.

Such limits on input signals are easily introduced by simply adding assumptions to the specification statement. We define a realisation step to allow this as follows.

Add assumption

$\vec{x} : [A, E]$ is replaced by $\vec{x} : [A \wedge B, E]$

For example, let $MaxVolts : Volts$ denote the maximum voltage which all of the components can tolerate. The necessary limit on the input signal is captured by the following schema.

VoltageLimits

$in : \mathbb{T} \rightsquigarrow Volts$

$[\![in \leq MaxVolts]\!] \subseteq [\![true]\!]$

Using the realisation step above, we have

$out : [Commands, SampleAndHold]$

is replaced by

$out : [Commands \wedge VoltageLimits, SampleAndHold]$.

Associated with each realisation step is a property transformation rule. This allows us to derive properties of the new specification from those proved for the original specification. For **Add Assumption**, the property transformation rule is:

If P is a property of $\vec{x} : [A, E]$
then $B \Rightarrow P$ is a property of $\vec{x} : [A \wedge B, E]$.

Proof: If P is a property of $\vec{x} : [A, E]$, it can be expressed in the form $S \Rightarrow T$ such that $S \Rightarrow A$ and $S \wedge E \Rightarrow T$.
Therefore, since $S \wedge B \Rightarrow A \wedge B$,

$$\begin{aligned} & S \wedge B \Rightarrow T \\ & \equiv B \Rightarrow (S \Rightarrow T) \\ & \equiv B \Rightarrow P \end{aligned}$$

is a property of $\vec{x} : [A \wedge B, E]$. $\square$

Such property transformation rules allows us to perform analysis and reasoning in the original ideal specification and transfer the results to the realisation of the specification in a straight-forward manner.

4 Timing errors

The implementation of Figure 3 also introduces a number of time delays including an acquisition time Ta, aperture time Tp and a settling time Ts [11, Section 2.4]. These are illustrated in Figure 4 which shows a typical output signal. The errors in the output signal itself are discussed in Section 5.

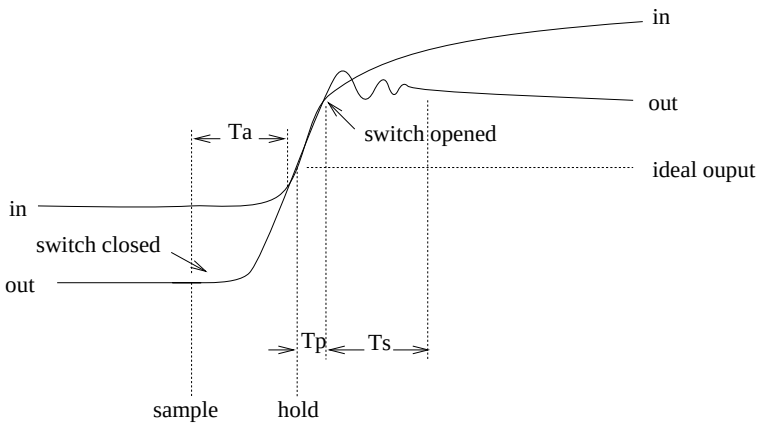

Figure 4: Sample-and-hold behaviour with time delays

4.1 Acquisition time

The acquisition time is the time from when the sample command is given until the output signal is within a specified error of the input signal. This time includes the time for the switch to be closed, and the time for the capacitor to charge up. The rate at which the capacitor charges depends on both its value and that of the resistor [17, Chapter 9].

It is necessary that a hold command does not occur before the acquisition time is complete. Hence, hold commands must be separated from sample commands by at least the maximum acquisition time.

Let $ClosingTime : \mathbb{T}$ denote the maximum time for the switch to close upon receiving the sample command. Let $ChargingTime : \mathbb{T}$ denote the maximum time for the capacitor to charge to within the specified error band of $MaxVolts$. (The actual charging time will depend on the precise values of the resistor and capacitor. The maximum charging time can be calculated by considering the specified tolerances on their values.) The separation between sample and hold commands is specified by the following schema.

$$
\begin{array}{|l}
\hline
CommandSeparation \\
sample, hold : \mathbb{T} \rightarrow \mathbb{B} \\
\hline
[\!| sample(\alpha) \land hold(\omega) |\!] \subseteq [\!| \delta \geq ClosingTime + ChargingTime |\!] \\
\hline
\end{array}
$$

Let $AcceptableInputs \mathrel{\widehat{=}} Commands \land VoltageLimits \land CommandSeparation$. Using **Add assumption** of Section 3, we can transform our specification of the sample-and-hold device as follows.

$out : [Commands \land VoltageLimits, SampleAndHold]$

is replaced by

$out : [AcceptableInputs, SampleAndHold]$

4.2 Aperture time

The aperture time is the time for the switch to open after the hold command is received. For a given switch (e.g., a transistor) this time will not be constant [11, Section 2.4].

It is necessary for our specification to reflect the fact that the output is charged to the value of the input after the aperture time and not at the application of the hold command. Such a requirement can be introduced into the specification by the introduction of a specification statement which constructs the perceived input, i.e., the delayed input as perceived by the ideal sample-and-hold device, from the actual input, and pipes this input to the original specification.

We define a realisation step to allow this as follows ($\vec{u}$ denotes a subset of the input variables of $\vec{x} : [A, E]$).

Modify input

$\vec{x} : [A, E]$ is replaced by $|[\, \vec{u} : [true, F] \gg \vec{x} : [A, E] \backslash \vec{u} \,]|$

Note that this step does not maintain the same names in the interface of the specification ($\vec{u}$ are no longer inputs). This is not necessary during realisation but can be achieved, if desired, using renaming.

Let $Skew : \mathbb{T}$ be the average aperture time of the switch and $Jitter : \mathbb{T}$ be half of the range in which the aperture time falls. The relationship between the actual command $hold_a$ and the perceived (ideal) command $hold$, after the delay, is captured by the following schema.

DelayedHold

$$hold_a, hold : \mathbb{T} \rightarrow \mathbb{B}$$

$$\begin{array}{l} \exists f : \mathbb{T} \rightarrowtail \mathbb{T} \bullet \\ \quad (\forall t : \mathbb{T} \bullet f(t) \in t + (Skew \pm Jitter)) \\ \quad hold_a = hold \circ f \end{array}$$

Using the realisation step above our specification is transformed as follows.

$out : [AcceptableInputs, SampleAndHold]$

is replaced by

$$|[\, hold : [true, DelayedHold] \\ \quad \gg out : [AcceptableInputs, SampleAndHold] \backslash \{hold\} \,]|$$

As with the **Add assumption** step, properties of the new specification can be derived from those of the original specification:

If P is a property of $\vec{x} : [A, E]$
then $(\forall \vec{u} \bullet F \Rightarrow A) \Rightarrow (\exists \vec{u} \bullet F \wedge P)$ is a property of

$|[\, \vec{u} : [true, F] \gg \vec{x} : [A, E] \backslash \vec{u}\,]|$.

Proof: If P is a property of $\vec{x} : [A, E]$, it can be expressed in the form $S \Rightarrow T$ such that $S \Rightarrow A$ and $S \wedge E \Rightarrow T$.

$$
\begin{aligned}
&|[\, \vec{u} : [true, F] \gg \vec{x} : [A, E] \backslash \vec{u}\,]| \\
&= |[\, \vec{u}, \vec{x} : [\forall\, \vec{u} \bullet F \Rightarrow A, F \wedge E] \backslash \vec{u}\,]| \\
&= \vec{x} : [\forall\, \vec{u} \bullet F \Rightarrow A, \exists\, \vec{u} \bullet F \wedge E]
\end{aligned}
$$

Since $S \wedge E \Rightarrow T \equiv E \Rightarrow (S \Rightarrow T)$,

$$
\begin{aligned}
&\exists\, \vec{u} \bullet F \wedge E \\
&\Rrightarrow \exists\, \vec{u} \bullet F \wedge (S \Rightarrow T) \\
&\equiv \exists\, \vec{u} \bullet F \wedge P.
\end{aligned}
$$

Therefore,

$$(\forall\, \vec{u} \bullet F \Rightarrow A) \Rightarrow (\exists\, \vec{u} \bullet F \wedge P)$$

is a property of $|[\, \vec{u} : [true, F] \gg \vec{x} : [A, E] \backslash \vec{u}\,]|$. $\square$

4.3 Settling time

The settling time is the time for any transient oscillations to settle to within a certain percent of the expected output value. Such oscillations occur due to the electronic switch not forming a perfect open circuit.

The output is only required to be equal to its expected value after the settling time. Such a requirement can be introduced into the specification by the introduction of a specification statement which constructs the actual output from the output of the original specification.

We define a realisation step to allow this as follows.

Modify output

$\vec{x} : [A, E]$ is replaced by $|[\, \vec{x} : [A, E] \gg \vec{x_a} : [true, F] \backslash \vec{x}\,]|$

Let $SettlingTime : \mathbb{T}$ denote the maximum settling time. The relationship between the actual output out_a and the ideal output out is captured by the following schema.

SettledOutput

$$
\begin{aligned}
&out : \mathbb{T} \rightarrow Volts \\
&out_a : \mathbb{T} \rightsquigarrow Volts \\
&\hline
&[\![\, out = out(\alpha) \wedge \delta > SettlingTime \,]\!] \subseteq \\
&\qquad [\![\, \delta = SettlingTime \,]\!] \,;\, [\![\, out_a = out \,]\!]
\end{aligned}
$$

Using the realisation step above our specification is transformed as follows.

$$|[\mathit{hold} : [\mathit{true}, \mathit{DelayedHold}]$$
$$\gg \mathit{out} : [\mathit{AcceptableInputs}, \mathit{SampleAndHold}] \backslash \{\mathit{hold}\}\,]|$$

is replaced by

$$|[\,|[\mathit{hold} : [\mathit{true}, \mathit{DelayedHold}]$$
$$\gg \mathit{out} : [\mathit{AcceptableInputs}, \mathit{SampleAndHold}] \backslash \{\mathit{hold}\}\,]|$$
$$\gg \mathit{out}_a : [\mathit{true}, \mathit{SettledOutput}] \backslash \{\mathit{out}\}\,]|$$

Since *DelayedHold* cannot refer to *out* and *SettledOutput* cannot refer to *hold*, this is equal to

$$|[\mathit{hold} : [\mathit{true}, \mathit{DelayedHold}]$$
$$\gg \mathit{out} : [\mathit{AcceptableInputs}, \mathit{SampleAndHold}]$$
$$\gg \mathit{out}_a : [\mathit{true}, \mathit{SettledOutput}] \backslash \{\mathit{hold}, \mathit{out}\}\,]|.$$

Hence, the realisation steps are associative and may be applied in any order.

The property transformation rule for **Modify output** is:

If P is a property of $\vec{x} : [A, E]$
then $\exists\, \vec{x} \bullet P \wedge (A \Rightarrow F)$ is a property of
$|[\vec{x} : [A, E] \gg \vec{x_a} : [\mathit{true}, F] \backslash \vec{x}\,]|$.

Proof: If P is a property of $\vec{x} : [A, E]$, it can be expressed in the form $S \Rightarrow T$ such that $S \Rightarrow A$ and $S \wedge E \Rightarrow T$.

$$|[\vec{x} : [A, E] \gg \vec{x_a} : [\mathit{true}, F] \backslash \vec{x}\,]|$$
$$= |[\vec{x}, \vec{x_a} : [A, E \wedge F] \backslash \vec{x}\,]|$$
$$= \vec{x_a} : [A, \exists\, \vec{x} \bullet E \wedge F]$$

Since $S \wedge (\exists\, \vec{x} \bullet E) \Rightarrow (\exists\, \vec{x} \bullet T)$ and the variables $\vec{x}$ do not occur free in S (since S describes constraints on the inputs only),

$$S \Rightarrow \exists\, \vec{x} \bullet T$$
$$\equiv \exists\, \vec{x} \bullet S \Rightarrow T$$
$$\equiv \exists\, \vec{x} \bullet P$$

is a property of $|[\vec{x} : [A, E] \gg \vec{x_a} : [\mathit{true}, F] \backslash \vec{x}\,]|$.
Also, since $A \wedge (\exists\, \vec{x} \bullet F) \Rightarrow (\exists\, \vec{x} \bullet F)$ and the variables $\vec{x}$ do not occur free in A,

$$A \Rightarrow \exists\, \vec{x} \bullet F$$
$$\equiv \exists\, \vec{x} \bullet A \Rightarrow F$$

is a property of $|[\vec{x} : [A, E] \gg \vec{x_a} : [\mathit{true}, F] \backslash \vec{x}\,]|$.
Hence,

$$\exists\, \vec{x} \bullet P \wedge (A \Rightarrow F)$$

is a property of $|[\vec{x} : [A, E] \gg \vec{x_a} : [\mathit{true}, F] \backslash \vec{x}\,]|$. $\square$

5 Signal errors

As well as timing errors, the implementation of Figure 3 introduces errors in the output signal. In particular, after the hold command is applied, the voltage across the capacitor may decay slightly due to leakage currents through the electronic switch. This is known as hold mode droop [11, Section 2.4].

Let $ReadingTime : \mathbb{T}$ denote the time in which the output of the sample-and-hold device is expected to be read. In digital systems, where the sample and hold commands are triggered by a periodic clock, this time may correspond to the time before the next sample command occurs. Let $Error : Volts$ denote the maximum deviation in the held output signal due to hold mode droop during $ReadingTime$.

The relationship between the output with and without hold mode droop, out_b and out_a respectively, is given by the following schema. ($[\![v \ldots w]\!]$ is the set of real numbers between v and w inclusive.)

$$
\begin{array}{|l}
\hline
HoldModeDroop \\
out_b, out_a : \mathbb{T} \rightsquigarrow Volts \\
\hline
[\![out_a = out_a(\alpha) \land SettlingTime < \delta < ReadingTime]\!] \subseteq \\
\quad [\![\delta = SettlingTime]\!] ; [\![out_b \in [\![out_a - Error \ldots out_a]\!]]\!] \\
\hline
\end{array}
$$

Note that a more accurate output could be specified which models the error increasing as time progresses. For our purposes, however, the above suffices.

Using the **Modify output** step of Section 4.3, we can transform our specification of the sample-and-hold device as follows.

$$
\begin{array}{l}
|[hold : [true, DelayedHold] \\
\quad \gg out : [AcceptableInputs, SampleAndHold] \\
\quad \gg out_a : [true, SettledOutput] \backslash \{ hold, out \}]|
\end{array}
$$

is replaced by

$$
\begin{array}{l}
|[\,|[hold : [true, DelayedHold] \\
\quad \gg out : [AcceptableInputs, SampleAndHold] \\
\quad \gg out_a : [true, SettledOutput] \backslash \{ hold, out \}]| \\
\quad \gg out_b : [true, HoldModeDroop] \backslash \{ out_a \}]|
\end{array}
$$

Due to the associativity of the steps, this is equal to the following.

$$
\begin{array}{l}
|[hold : [true, DelayedHold] \\
\quad \gg out : [AcceptableInputs, SampleAndHold] \\
\quad \gg out_a : [true, SettledOutput] \\
\quad \gg out_b : [true, HoldModeDroop] \backslash \{ hold, out, out_a \}]|
\end{array}
$$

6 Conclusions and related work

In this paper, we have introduced the notion of realisation of real-time specifications. Realisation is an approach to specification development which involves transforming the specification to include the physical limitations, and timing and signalling errors inherent in an implementation. This enables top-level specifications to ignore these details resulting in clearer, more easily analysed, descriptions.

We presented a small number of realisation steps which are general enough to allow most kinds of limitations and errors to be introduced. This generality, however, means that these steps must be used with care. It is the responsibility of the specifier to ensure that some approximation to the original specification is maintained in a realisation step. This could be validated, for example, by examination of the derived properties of the transformed specification.

More specific realisation steps would remove some of this responsibility from the specifier. For example, a general schema based on the schema *DelayedHold* could be used as part of a specific step for introducing time delays in terms of skew and jitter.

As mentioned previously, the introduction of physical limitations and errors into specifications was considered by Hayes [9]. This work was followed by that of Hayes and Sanders [10] in which a general approach (using piping) was presented for changing the input and output representation of Z specifications. This work although similar to ours, focuses on specification issues such as clarity and reuse, as opposed to realisation. Hence, it does not deal with the issue of how properties are transformed.

A paper by Boiten and Derrick [2], on the other hand, uses an approach similar to that of Hayes and Sanders to define a notion of Z refinement which allows changes to the inputs and outputs of specifications. Since refinement, as opposed to realisation, is the goal of this work, however, these changes are limited. It is not, therefore, as general as our approach. Similary, approaches to *conditional refinement* [4, 8], which allow assumptions to be added to specifications, are also not as general as our approach.

The work closest in aims to our paper is that of Banach and Poppleton [1] whose notion of *retrenchment* is identical to our notion of realisation. There is, however, no mathematical basis for this approach and hence no way to derive properties of the more concrete specifications.

In addition to the differences already noted with our work, none of the above-mentioned approaches uses a specification notation suited to our application domain of real-time systems.

Acknowledgements

Thanks to Colin Fidge and Ian Hayes for many helpful comments on this paper. This work is funded by Australian Research Council (ARC) grant number A49801500: *A Unified Formalism for Concurrent Real-Time Software Development.*

References

1. R. Banach and M. Poppleton. Retrenchment: An engineering variation on refinement. In D. Bert, editor, *Proceedings of B-98*, volume 1393 of *LNCS*, pages 129–147. Springer-Verlag, 1998.
2. E.A. Boiten and J. Derrick. IO-refinement in Z. In *3rd BCS-FACS Northern Formal Methods Workshop*, Electronic Workshops in Computing. Springer-Verlag, 1998.
3. M. Broy. Refinement of time. In *AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'97)*, volume 1231 of *LNCS*. Springer-Verlag, 1997.
4. W. Chen and J.T. Udding. Towards a calculus of data refinement. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction (MPC'89)*, volume 375 of *Lecture Notes in Computer Science*, pages 197–218. Springer-Verlag, 1989.
5. K. Duddy, L. Everett, C. Millerchip, B.P. Mahony, and I.J. Hayes. Z-based notation for the specification of timing properties. Department of Computer Science, University of Queensland, June 1995.
6. C.J. Fidge, I.J. Hayes, and B.P. Mahony. Differential and integral calculus in Z. In J. Staples, M.G. Hinchey, and Shaoying Liu, editors, *IEEE International Conference on Formal Engineering Methods (ICFEM '98)*, pages 64–73. IEEE Computer Society, 1998.
7. C.J. Fidge, I.J. Hayes, A.P. Martin, and A.K. Wabenhorst. A set-theoretic model for real-time specification and reasoning. In J. Jeuring, editor, *Mathematics of Program Construction (MPC'98)*, volume 1422 of *Lecture Notes in Computer Science*, pages 188–206. Springer-Verlag, 1998.
8. A. Gravell. Specialising abstract programs. In J.M. Morris and R.C. Shaw, editors, *4th Refinement Workshop*, Workshops in Computing, pages 34–50. Springer-Verlag, 1991.
9. I.J. Hayes. Specifying physical limitations: A case study of an oscilloscope. Technical Report 167, Department of Computer Science, University of Queensland, 1990. Revised 1993.
10. I.J. Hayes and J.W. Sanders. Specification by interface separation. *Formal Aspects of Computing*, 7(4):430–439, 1995.
11. B.C. Kuo. *Digital Control Systems*. HRW Series in Electrical and Computer Engineering. CBS Publishing Japan, 1980.
12. B.P. Mahony. *The Specification and Refinement of Timed Processes*. PhD thesis, Department of Computer Science, University of Queensland, 1992.
13. B.P. Mahony and I.J. Hayes. A case-study in timed refinement: A mine pump. *IEEE Transactions on Software Engineering*, 18(9):817–826, September 1992.
14. C. Millerchip, B.P. Mahony, and I.J. Hayes. The generic problem competition: A whole system specification of the boiler system. Software Verification Research Centre, University of Queensland, June 1993.
15. D. Scholefield and H. Zedan. A standard for finite TAM. Technical Report YCS 206, Department of Computer Science, University of York, 1993.
16. D. Scholefield, H. Zedan, and He Jifeng. A specification-oriented semantics for the refinement of real-time systems. *Theoretical Computer Science*, 131:219–241, 1994.
17. R.J. Smith. *Circuits, Devices and Systems*. John Wiley and Sons, 1976.
18. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. International Series in Computer Science. Prentice-Hall, 1996.
19. Zhou Chaochen and C.A.R. Hoare and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–271, December 1991.