A Systematic Design of Real-time Systems Using Duration Calculus

Do Van Nhon and Dang Van Hung
The United Nations University
International Institute for Software Technology
UNU/IIST, P.O.Box 3058, Macau

ABSTRACT

We present a systematic way for the design and verification of real-time systems using Duration Calculus. We formalise the synchronous interface between the components of the systems as duration calculus formulas expressing the approximation between state variables. Preliminary Designs are then derived from the safety requirements and the interface. Functionality requirements are incorporated to be consistent with the Preliminary Designs. The verification presented in this paper can be reused for other systems as well like Biphase Mark Protocol.

Keywords: Formal Methods, Duration Calculus, Real-time Control Systems, Biphase Mark Protocol

1. INTRODUCTION

Synchronous systems are distributed systems in which there are known upper bound on the time it takes for a message sent by one component to be received by another, and time it takes for a component to perform certain operation. A critical real-time system such as a software embedded control system can be implemented as a synchronous system. Each component of the system uses its own local state variables, and other components can learn about its behaviour with some non-zero delay. Commands given by one component to another can be performed with some delay as well. Since a critical real-time system has to satisfy some real-time requirements, the delays allowed by a synchronous system to implement it should not be too long. It is worth considering the common relation among the delays and the parameters of the real-time requirements of the system that often encountered in practice. For such kind of systems, there is no common clock. So, the continuous time is most natural and suitable for them. We found that the duration calculus (DC) with continuous time and (relative) complete prove system is among the best formalism to specify the relationship between components of synchronous systems. Since in DC a system is modelled by a set of state variables which are $\{0, 1\}$-valued functions of time, the interface between components is modelled easily by the approximation of these kind of functions.

In this paper we show a class of DC formulas which are commonly used to express the approximation between state variables, and show which approximation level is needed to satisfy a certain real-time requirement. Via a toy example, we also present a systematic way to develop a critical real-time system from its requirements. First, we have to give an abstract state model for the main components that are related to our safety requirements. Then, we formalised the safety requirements as DC formulas over that state model together with the relationship between states and the assumption about the environment. The specification of the safety should be as weak as possible to prevent the inconsistency when we incorporate the functionality requirement. Preliminary design decisions then will be derived from those assumptions and safety requirements. When the functionality requirement is taken into account, there may be a trade-off between the safety, feasibility and the efficiency. We found that the formal verification in this paper captures a common form in the design process of real-time systems, and can be considered as a theorem to be used later in different development. We consider the biphase mark protocol that has been well known in the literature and found that our model of verification works well for this case. In comparison to the methods in the literature [2], our method can model the protocol in more natural way with more detailed physical assumptions and higher accuracy. The result of the verification can help to choose the optimal parameters for the protocol.

The paper is organised as follows. The next section is an introduction of notations in Duration Calculus that will be used in the paper. In Section 3, we present the design and verification technique via a toy example of a motor control system. Section 4 of the paper is to show how our verification procedure could be applied to another nontrivial example. The last section is the conclusion of the paper.

2. DURATION CALCULUS: A BRIEF SUMMARY

In this section, we give a brief summary of Duration Calculus which will be used as the specification language for the design of real-time systems in this paper. For more details, readers are referred to [1].

Time in DC is the set R^+ of non-negative real numbers. For $t, t' \in R^+, t \leq t'$, $[t, t']$ denotes the time interval from t to t'.

We assume a finite set E of state variables. E includes the Boolean constants 0 and 1 denoting *false* and *true* respectively. A state variable P is interpreted as a function P_I (or just P when I is understood) from *Time* to $\{0, 1\}$. $P_I(t) = 1$ means that state P is present at time instant t, and $P_I(t) = 0$ means that state P is not present at time instant t under interpretation I. We assume that a state has finite variability in a finite time interval. A Boolean state expression is interpreted as a function which is defined by the interpretations for the state variables and Boolean operators.

For an arbitrary state variable P, its duration is denoted by $\int P$. Given an interpretation I of states and a time interval

$[t,t'])$, duration $\int P$ is interpreted as the accumulated length of time within the interval at which P is present, i.e., $\int_t^{t'} P_I(t)dt$. Therefore, $\int 1$ always gives the length of the reference interval and is denoted by ℓ.

The set of primitive duration terms consists of constants (real numbers) and durations of state expressions. A duration term is defined either as a primitive term or as a combination of primitive terms using arithmetic operators.

A primitive duration formula is an expression formed from terms by using the usual relational operations on the reals, such as equality = and inequality $<$. So, a primitive duration formula is of the form $R(t_1,\ldots,t_m)$, where R is a relation, t_i's are term. Duration formulas are generated by the grammar:

$$\Phi =_{\text{def}} A \mid \Phi \vee \Phi \mid \neg\Phi \mid \Phi;\Phi$$

where A stands for a primitive duration formula.

A duration formula D is satisfied by an interpretation I in interval $[t',t'']$, denoted by $I,[t',t''] \models D$, just when it evaluates to true for that interpretation over that time interval. Given an interpretation I, a primitive formula $R(t_1,\ldots,t_m)$ is evaluated to true for a time interval $[t',t'']$ iff the interpreted value of $t_1,\ldots,t_m$ over $[t',t'']$ satisfies the relation R. The chop-formula $D_1;D_2$ is true iff there exists a time point t such that $t' \le t \le t''$ and D_1 and D_2 are true for $[t',t]$ and $[t,t'']$, respectively.

We give now shorthands for some duration formulas which are often used. For an arbitrary state P, $\lceil P \rceil$ stands for $(\int P = \ell) \wedge (\ell > 0)$. This means that P holds (almost) everywhere in a non-point interval. We use $\lceil\,\rceil$ to denote the predicate which is true only for point intervals. Modalities $\Diamond, \Box$ are defined as: $\Diamond D = true;D;true$, $\Box D = \neg\Diamond\neg D$. This means that $\Diamond D$ is true for an interval iff D holds for some subinterval of it, and $\Box D$ is true for an interval iff D holds for all subintervals of it.

DC has a set of axioms about states and rules which is (relatively) complete. The readers are referred to [1] for the proof system of DC.

3. REAL-TIME SYSTEM DESIGN USING DC

The architecture of a real-time control system can be represented by the diagram in Figure 1. There are two parts in the system: discrete and continuous. Our goal is to derive a design for the discrete part from the requirements and assumptions for the motor control system presented below. We specify the system in Duration Calculus, and prove the correctness of the design formally using proof system of DC. The proof can be checked by DC proof checkers.

Specification for Motor Control System - a Toy Example

Suppose that a motor control system has the following safety requirements:

Req 1: The motor cannot be turned on when the temperature is high (above k C degree).

Req 2: When the motor is running and the temperature is high, the motor should be turned off within a time units.

Req 3: The motor cannot be turned on without having been off for at least b time units.

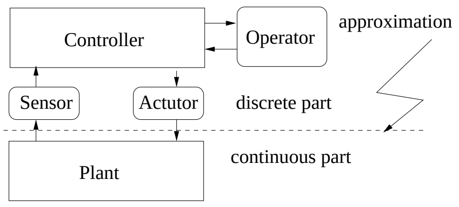

Figure 1. Architecture of Real-Time Control System

The state model for the motor and the control system consists of Boolean-valued functions expressed by state variables in DC. States related to the motor are “high temperature” and “running motor”. These two states are expressed by the state variables $High\_T$ and M_on as follows:

$$High\_T(t) =_{\text{def}} \begin{cases} 1 & \text{the temperature is high at time } t \\ 0 & \text{the temperature is low at time } t \end{cases}$$

$$M\_on(t) =_{\text{def}} \begin{cases} 1 & \text{the motor is running at time } t \\ 0 & \text{the motor is not running at time } t \end{cases}$$

So, the event ”turn on (off) motor” means a change of the state variable M_on from 0 to 1 (1 to 0), and the event ”temperature changes from low to high (high to low)” means a change of the state variable $High\_T$ from 0 to 1 (1 to 0). Let $M\_off =_{\text{def}} \neg M\_on$. The safety requirements are specified in DC as follows:

Safety (a): Motor cannot be turned on in high temperature:
$\Box(\lceil High\_T \rceil \Rightarrow \Box\neg(\lceil M\_off \rceil;\lceil M\_on \rceil))$

Safety (b): Motor should be off within a time units if the temperature is high:
$\Box(\lceil M\_on \wedge High\_T \rceil; \ell > a \Rightarrow \ell \le a; \lceil M\_off \rceil; True)$

Safety (c): Motor cannot be on without having been off for at least b time units:
$\Box(\lceil M\_on \rceil;\lceil M\_off \rceil;\lceil M\_on \rceil \Rightarrow \ell \ge b)$

It is natural to assume the stability of the operation for the motor which says that the motor must stay in each state (on or off) for at least δ time units each time it enters. Without this assumption, there is no way for the controller to control the operation of motor when there is a non-zero delay for a controller command to be effective, and we cannot make the system to satisfy the safety when the motor can change from off to on and from on to off freely. This assumption is specified by:

$$(\mathcal{A}1) \begin{cases} \Box(\lceil M\_on \rceil;\lceil M\_off \rceil;\lceil M\_on \rceil \Rightarrow \ell \ge \delta)\ , \\ \Box(\lceil M\_off \rceil;\lceil M\_on \rceil;\lceil M\_off \rceil \Rightarrow \ell \ge \delta)\ . \end{cases}$$

Since it takes time for the sensor to detect a change of the temperature from high to low or from low to high, and since there is a delay for the controller to turn the motor on or off, the controller must detect the high temperature at a certain time before this happens so that it can conduct the motor timely and safely. Therefore, we need the assumption that there is a threshold ”critical temperature” for which it takes a certain amount of time for the temperature to become high from non-critical. Without this assumption, the controller can not guarantee the Safety (a) because it can make the command ”turn on” when the temperature is not high but by the delay when the motor change from off to on the temperature is high. The critical temperature is expressed by state variable Cri_T:

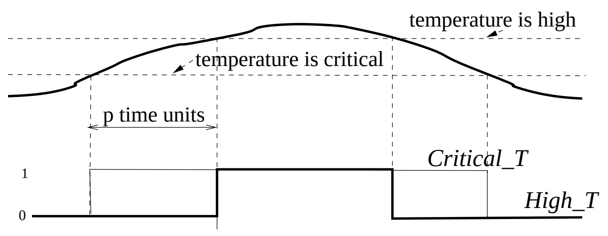

Figure 2. States $High_T$ and $Critical_T$

$$Cri\_T(t) = \begin{cases} 1 & \text{the temperature is critical at } t \\ 0 & \text{otherwise} \end{cases}$$

Let p be the lower bound of time for the temperature to become high from non-critical. So, to become high from non-critical, the temperature must be in critical state for at least p time units (see Figure 2). This assumption is specified by the DC formulae:

$(\mathcal{A}2)$ $\lceil High\_T \rceil \Rightarrow \lceil Cri\_T \rceil$

$(\mathcal{A}3)$ $\lceil \neg Cri\_T \rceil ; True; \lceil High\_T \rceil$
$\Rightarrow \lceil \neg Cri\_T \rceil ; \lceil Cri\_T \rceil \wedge \ell \geq p; True$

Furthermore, we assume that:

- It takes h time units for the controller to detect a change of the temperature from non-critical to critical.
- It takes d time units for the controller to turn the motor on or off.

We say that h is the "sampling time" of the controller and d is the "delay time" for the controller to control the motor. These assumptions express the approximation between discrete and continuous states. To specify these assumptions, we define two state variables for the controller: variable Cmd_on_c expressing the state of the switch and variable Cri_T_c expressing the sampled result of the state Cri_T.

$$Cmd\_on_c(t) = \begin{cases} 1 & \text{the switch is on at time } t \\ 0 & \text{the switch is off at time } t \end{cases}$$

$$Cri\_T_c(t) = \begin{cases} 1 & \text{controller knows that} \\ & \text{temperature is critical at } t \\ 0 & \text{otherwise} \end{cases}$$

Let $Cmd\_off_c =_{\text{def}} \neg Cmd\_on_c$. We specify the assumptions by the following DC formulae:

$(\mathcal{A}4)$ $\lceil Cmd\_off_c \rceil \wedge \ell \geq d \Rightarrow \ell < d; \lceil M\_off \rceil$
$(\mathcal{A}5)$ $\lceil Cmd\_on_c \rceil \wedge \ell \geq d \Rightarrow \ell < d; \lceil M\_on \rceil$
$(\mathcal{A}6)$ $\lceil Cri\_T \rceil \wedge \ell > h \Rightarrow \ell \leq h; \lceil Cri\_T_c \rceil$
$(\mathcal{A}7)$ $\lceil \neg Cri\_T \rceil \wedge \ell > h \Rightarrow \ell \leq h; \lceil \neg Cri\_T_c \rceil$

To guarantee that the operation of the motor is controlled by the controller, the delay d should not be greater than 2δ (δ is the stable time of the motor). This implies that every change of the motor is caused by the command. Besides, for the controller to be able to control the motor timely and safely in the case that the temperature changes from non-critical to high, the time for temperature to change from non-critical to high must be greater than the sum of sampling time and delay time. Therefore, we assume that the parameters δ, p, h and d satisfy the condition:

$$(\mathcal{C}1) \begin{cases} d \leq 2\delta \\ p \geq h + d \end{cases}$$

A design of the controller to satisfy the safety requirements (a), (b) and (c)is specified by DC formulae over discrete state variables Cri_T_c and Cmd_on_c:

$(\mathcal{D}1)$ While Cri_T_c is 1, the controller should not switch on the motor.
$\lceil Cri\_T_c \rceil \Rightarrow \neg \Diamond(\lceil Cmd\_off_c \rceil ; \lceil Cmd\_on_c \rceil)$

$(\mathcal{D}2)$ If the switch is on while Cri_T_c is 1, the switch should be off within a' time units.
$\lceil Cri\_T_c \wedge Cmd\_on_c \rceil ; \ell > a' \Rightarrow$
$\ell \leq a'; \lceil Cmd\_off_c \rceil ; True$

$(\mathcal{D}3)$ The stable time of the switch at the state "off" is at least b' time units.
$\lceil Cmd\_on_c \rceil ; \lceil Cmd\_off_c \rceil ; \lceil Cmd\_on_c \rceil \Rightarrow \ell \geq b'$

$(\mathcal{D}3')$ The stable time of the switch at the state "on" is at least d time units.
$\lceil Cmd\_off_c \rceil ; \lceil Cmd\_on_c \rceil ; \lceil Cmd\_off_c \rceil \Rightarrow \ell \geq d$

We need the design $(\mathcal{D}3')$ because of the delay d for the controller to turn the motor on or off. Without it, the motor could not react to the controller's command. The parameters must satisfy certain constraints for the design to be correct. To avoid the obvious conflict between designs and safety, we should have:

$(\mathcal{C}2)$ $a' \geq d \wedge a' + d \leq a \wedge b' \geq b + d$

Apart from the safety requirement, the system is also required to specify the functionality one. It requires the reaction of the system to a request for turning on the motor from the operator. To express the functionality requirement We define a state variable for expressing the operator request as follows:

$$Request\_on_c(t) = \begin{cases} 1 & \text{operator requests motor on at } t \\ 0 & \text{otherwise} \end{cases}$$

Of course, the system cannot always turn on the motor when the request is "on" because it must guarantee the safety requirements. It cannot turn on the motor immediately because of the delay. By the safety (a), the request should be in non-critical temperature. So, without taking into account the efficiency we can formulate the functionality requirement as follows: if the operator requests to turn on the motor while the temperature is non-critical, and suppose the request lasts for long enough time (f time units), then the motor will be turned on after a certain time. This statement is specified by:

$$(\mathcal{F}) \quad \Box \begin{pmatrix} \lceil Request\_on_c \wedge \neg Cri\_T \rceil \wedge \ell > f \\ \Rightarrow \ell \leq f; \lceil M\_on \rceil \end{pmatrix}$$

Since the functionality requirement must be consistent to the safety requirements, the parameter f must satisfy certain constraints. To determine constraints on parameter f in the functionality requirement, we consider the worst case of the system illustrated in Figure 3. In the interval $[t, t']$ we have $\lceil Request\_on_c \wedge \neg Cri\_T \rceil$. Because of sampling time h in assumptions $(\mathcal{A}6)$ and $(\mathcal{A}7)$, the controller does not know that the temperature is non-critical in the subinterval $[t, t1]$. In the subinterval $[t1, t2]$ the controller knows that the request is on and the temperature is non-critical, but it cannot turn Cmd_on "on" because Cmd_on should be "off" in this subinterval by the design $(\mathcal{D}2)$. By the design $(\mathcal{D}3)$, Cmd_on must be "off" in subinterval $[t2, t3]$. Therefore, the controller can only turn Cmd_on

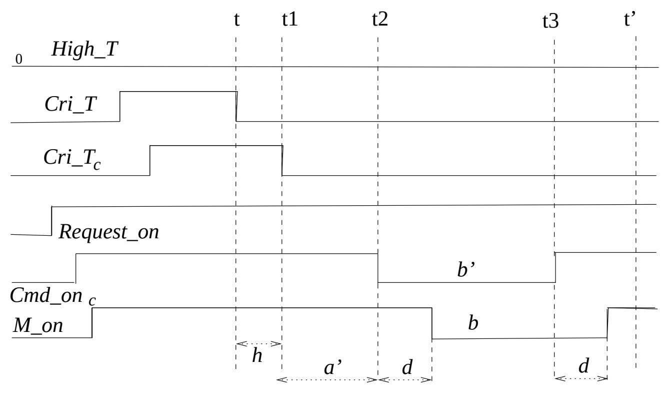

Figure 3. A State situation

on in the subinterval $[t3, t']$ to start the motor. However, by the delay d in the assumptions $(\mathcal{A}4)$ and $(\mathcal{A}5)$, the motor can be on after d time units from the beginning of the subinterval $[t3, t']$.

The above analysis shows that the controller cannot be sure whether the motor can be on within $(h + a' + b' + d)$ time units from the beginning of the interval $[t, t']$. This requires the parameter f in the requirement $(\mathcal{F})$ to be at least to $(h+a'+b'+d)$:

$(\mathcal{C}3)\ f \geq h + a' + b' + d$

for $(\mathcal{F})$ to be consistent with designs for safety requirements.

The design for functionality requirement must be as efficient as possible. This means that the controller must turn Cmd_on "on" as soon as possible when the operator requests to turn on the motor in case that the temperature is non-critical. Of course we can decide a design for the functionality requirement $(\mathcal{F})$ as:

$$\lceil Request\_on_c \wedge \neg Cri\_T_c \rceil \wedge \ell > h + a' + b' + d \\ \Rightarrow \ell \leq h + a' + b' + d; \lceil Cmd\_on_c \rceil$$

However, the above decision is not effective because in many cases the worst situation mentioned in Figure 3 does not happen and it does not need to wait so long time before switching Cmd_on to "on". To have a more effective design we use a discrete state variable $Possible_c$ for the controller to observe the situation (the index c to emphasise discrete property of the state variable). The intuitive meaning of $Possible_c$ is that it gives us the information about the time when the controller can switch the command to on.

From the above analysis we design for the state variable $Possible_c$ by the following DC formulae:

$$(\mathcal{D}4) \left\{ \begin{array}{ll} 1. & \lceil Cmd\_off_c \rceil; \lceil Cmd\_on_c \wedge \neg Cri\_T_c \rceil \\ & \quad \Rightarrow \lceil Cmd\_off_c \rceil; \lceil Possible_c \rceil \\ 2. & \lceil Cmd\_on_c \rceil \wedge (\lceil Cri\_T_c \rceil; true) \\ & \quad \Rightarrow \lceil \neg Possible_c \rceil \\ 3. & \lceil Cmd\_on_c \rceil; \lceil Cmd\_off_c \rceil \wedge \ell \leq b' \\ & \quad \Rightarrow \lceil Cmd\_on_c \rceil; \lceil \neg Possible_c \rceil \\ 4. & \lceil Cmd\_off_c \rceil \wedge \ell > b' \Rightarrow \ell = b'; \lceil Possible_c \rceil \end{array} \right.$$

Then we decide the design $(\mathcal{D}5)$ as follows:

$$(\mathcal{D}5) \left(\begin{array}{l} \lceil Request\_on_c \wedge \neg Cri\_T_c \rceil \\ \Rightarrow \Box(\lceil Possible_c \rceil; true \Rightarrow \lceil Cmd\_on_c \rceil) \end{array} \right)$$

Let

As $=_{\text{def}} \{(\mathcal{A}1), (\mathcal{A}2), (\mathcal{A}3), (\mathcal{A}4), (\mathcal{A}5), (\mathcal{A}6), (\mathcal{A}7)\}$

Des $=_{\text{def}} \{(\mathcal{D}1), (\mathcal{D}2), (\mathcal{D}3), (\mathcal{D}3'), (\mathcal{D}4), (\mathcal{D}5)\}$

Req $=_{\text{def}}$ *Safety(a)* $\wedge$ *Safety(b)* $\wedge$ *Safety(c)* $\wedge (\mathcal{F})$

$$\textbf{\textit{Init}} =_{\text{def}} \bigwedge \left\{ \begin{array}{l} \lceil M\_off \rceil \wedge \ell \geq \delta; true \\ \lceil Cmd\_off_c \rceil \wedge \ell \geq d; true \\ \lceil \neg Cri\_T \rceil; true \end{array} \right\}$$

The correctness of the design is formulated as:

Theorem 1 *Under the conditions* $(\mathcal{C}1)$, $(\mathcal{C}2)$ *and* $(\mathcal{C}3)$
Des, ***As*** $\vdash$ ***Init*** $\Rightarrow \Box$***Req***

The readers are referred to [4] for a formal proof of the theorem with the DC proof system.

3.2. Design Method

From the above example, we now propose a method for designing and verifying timed-synchronous systems using DC. The general idea of our method is that we model the system as a set of states, and divide the states into two parts: continuous part and discrete part. After formalising the requirements, the interface between two worlds (discrete and continuous) and the assumption over the states as DC formulas, we derive a design for the controller as a DC formulae over discrete variables (in the discrete world) that meets the requirements under the assumptions. Our design technique has the following steps:

Step 1: Determine states of the system and define the corresponding state variables. They are divided into two kinds: continuous and discrete.

Step 2: Specify the requirements over continuous states consisting of the safety requirements and the functionality requirement for the system as well as the assumptions about the environment.

Step 3: Describe the "interface" between continuous states and discrete states as the approximations of the continuous states and the discrete states derived from the time bounds of the synchronous distributed systems.

Step 4: Find (calculate) the specifications over discrete states such that these specifications satisfy the requirements under the assumptions and the interface.

Step 5: Optimisation using auxiliary states.

Verification and Design Rules Let us denote:

$switch(X, Y) =_{\text{def}} \lceil Y \wedge \neg X \rceil; \lceil X \rceil$
$X \rhd_\delta Y =_{\text{def}} \Box(\lceil X \rceil \wedge \ell \geq \delta \Rightarrow \ell < \delta; \lceil Y \rceil)$
$\delta\_stable(S) =_{\text{def}} \Box(\lceil \neg S \rceil; \lceil S \rceil; \lceil \neg S \rceil \Rightarrow \lceil \neg S \rceil; \lceil S \rceil \wedge \ell > \delta$

Below we give some rules that are useful for design and verification in cases there is a state to be controlled by another state with a delay. We assume that state M is controlled by state Cm with the delay τ, i.e. $Cm \rhd_\tau M$ and $\neg Cm \rhd_\tau \neg M$. Let

$init(X, a) =_{\text{def}} (\lceil X \rceil \vee \lceil \neg X \rceil) \wedge l \geq a; true$
$\Gamma =_{\text{def}} \{\delta\_stable(M), \delta\_stable(\neg M),$
$\qquad \tau\_stable(Cm), \tau\_stable(\neg Cm)\}$

Under the condition $\tau \leq 2.\delta$, we have:

Rule 1: $\Gamma \ \vdash$

$$\left(\begin{array}{l} ((\lceil M \rceil; true) \vee (\lceil \neg M \rceil \wedge \ell \geq \delta; true)) \wedge init(Cm, \tau) \\ \Rightarrow \Box(\lceil \neg M \rceil; \lceil M \rceil \Rightarrow \lceil \neg M \rceil; \lceil M \rceil \wedge (\lceil Cm \rceil; true)) \end{array} \right)$$

Rule 2: $\Gamma \ \vdash$

$$\left(\begin{array}{l}((\lceil\neg M\rceil;true)\vee(\lceil M\rceil\wedge\ell\geq\delta;true))\wedge init(Cm,\tau)\\ \Rightarrow\Box(\lceil M\rceil;\lceil\neg M\rceil\Rightarrow\lceil M\rceil;\lceil\neg M\rceil\wedge(\lceil\neg Cm\rceil;true))\end{array}\right)$$

Rule 3: $\Gamma\ \vdash$
$$\left(\begin{array}{l}init(M,\delta)\wedge init(Cm,\tau)\Rightarrow\\ \Box(\lceil\neg M\rceil;\lceil M\rceil;\lceil\neg M\rceil\Rightarrow\lceil\neg M\rceil;switch(\neg Cm,M);true)\end{array}\right)$$

Rule 4: $\Gamma\ \vdash$
$$\left(\begin{array}{l}init(M,\delta)\wedge init(Cm,\tau)\Rightarrow\\ \Box(\lceil M\rceil;\lceil\neg M\rceil;\lceil M\rceil\Rightarrow\lceil M\rceil;switch(Cm,\neg M);true)\end{array}\right)$$

Rules 3 and 4 say that every change in state M is caused by a change in state Cm if the states M and Cm are initiated long enough.

Rule 5: $\Gamma,\ (b+\tau)\_stable(\neg Cm)$
$$\vdash\ \bigwedge\left\{\begin{array}{l}\lceil\neg M\rceil\wedge\ell\geq\delta;true\\ init(Cm,\tau)\end{array}\right\}\Rightarrow b\_stable(\neg M)$$

Rule 6: $\Gamma,\ \ell=\tau;\lceil H\rceil\Rightarrow\neg\Diamond(\lceil\neg Cm\rceil;\lceil Cm\rceil)$
$$\vdash\ \bigwedge\left\{\begin{array}{l}\lceil\neg H\rceil\wedge l\geq\tau;true\\ init(M,\delta)\wedge init(Cm,\tau)\end{array}\right\}$$
$$\Rightarrow\Box(\lceil H\rceil\Rightarrow\neg\Diamond(\lceil\neg M\rceil;\lceil M\rceil))$$

Rule 7: Let $a>\tau$.
$$\left\{\begin{array}{l}\lceil H\rceil\Rightarrow\neg\Diamond(\lceil\neg Cm\rceil;\lceil Cm\rceil)\ ,\\ \lceil H\wedge Cm\rceil;\ell>a-\tau\Rightarrow\ell\leq a-\tau;\lceil\neg Cm\rceil;true\end{array}\right\}$$
$\vdash\ init(Cm,\tau)\Rightarrow\Box(\lceil H\wedge M\rceil;\ell>a\Rightarrow\ell\leq a;\lceil\neg M\rceil;true)$

Rule 8: Assume that $p>h$
$$\left\{\begin{array}{l}\lceil H\rceil\Rightarrow\lceil C\rceil,\\ \lceil\neg C\rceil;true;\lceil H\rceil\Rightarrow\lceil\neg C\rceil;\lceil C\rceil\wedge\ell\geq p;\lceil H\rceil,\\ C\ \rhd_h\ C_c,\\ \lceil C_c\rceil\Rightarrow\neg\Diamond(\lceil\neg Cm\rceil;\lceil Cm\rceil)\end{array}\right\}$$
$\vdash\ \lceil\neg C\rceil;true\Rightarrow\Box(\ell\leq p-h;\lceil H\rceil\Rightarrow\neg\Diamond(\lceil\neg Cm\rceil;\lceil Cm\rceil)$

Rules 5-8 are useful in the design process. Rule 5 gives a design over Cm to guarantee the stability of the state M. Rule 6 and 7 give a design step in which a design over states Cm and H will satisfy a property over states H and M under certain assumptions. In rule 8 we have a design over states C_c and Cm that satisfies a property related to state H under some assumptions (e.g. not to turn on the motor when the temperature is high).

4. BIPHASE MARK PROTOCOL

In this section, we apply our discretization technique to a real industrial example, Biphase Mark Protocol (BMP). There have been several papers presented methods for formal specification and verification of BMP (see, e.g. [3] and [5]). However, in [5] it does not give the concrete relations on parameters. In [3], the specification is over-specified because specifications is based on the receiver's clock (we referred to the private conversation between the author and Frits Vaandrager).

In this section we present a natural way to specify the Biphase Mark Protocol with more detailed physical assumptions and higher accuracy using the technique presented in section 2. Unlike [3], we use the sender's clock for reference. We give the concrete constraints on parameters. These constraints are useful for selecting the best values of the parameters in the design.

The BMP protocol encodes a bit as a cell consisting of a mark subcell and a code subcell. If the signal in the mark subcell is the same as the one in the code subcell then the information carried by that cell is 0. Otherwise, the information carried is 1. There is a phase reverse between two consecutive cells, i.e. the signal at the beginning of the following cell is held as the negation of the signal at the end of the previous cell. The receiver detects the beginning of a cell by recognising a change of the signals received, which is called an *edge*. The receiver, after having detected the beginning of a cell, skips some cycles called sampling distance and samples the signal. If the sampled signal is the same as the signal at the beginning of the cell, it decodes the cell as 0; otherwise it decodes the cell as 1. Let b be the length in time of a mark subcell, a the length in time of a code subcell and d the sample distance. We model the sender's signals by the state variable X and receiver's signals by the state variable Y. Since we use the sender's clock as the reference for time, the states X is discrete. Suppose that the signals sent are unbelief in r cycles ($r<a$ and $r<b$) after a change. Without loss of generality, we assume that the delay between the sender and the receiver is 0.

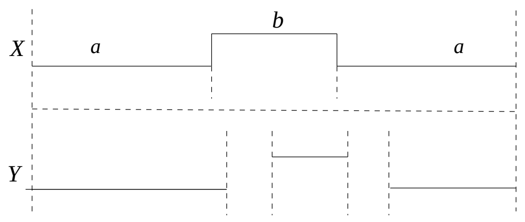

Figure 4. Sending the bit 1

There are 4 cases the cell encodes one bit depending on the values of the mark subcell and the code subcell. We consider the time interval with the length of $a+b+a$ consisting of a cell and a cycles before the cell when the sender sends bit 1. Figure 4 illustrates the case in which the mark subcell is 1 and the code subcell is 0.

Suppose the time length between two consecutive ticks of receiver's clock is Δ. The parameters b, a, r and Δ must satisfy certain constraints to guarantee conditions: (1) the receiver recognises the edges, and (2) the receiver samples the belief codes. By arithmetic calculations, we found that the constraints are:

$$(\mathcal{C}1)\left\{\begin{array}{l}b-r\geq 2\Delta\\ a-r\geq 2\Delta\end{array}\right.\quad\text{(for recognizing edge)}$$

$$(\mathcal{C}2)\left\{\begin{array}{l}d\geq\lceil(b+r)\theta\rceil+1\\ d\leq\lceil(b+a-r)\theta\rceil-3\end{array}\right.\quad\text{(for sampling belief code)}$$

where $\theta=\Delta^{-1}$ and $\lceil x\rceil\ =_{\text{def}}$ the least integer greater than or equal to x.

For each sequence of bits $\omega\in\{0,1\}^+$ we express the signals sent by a DC formula $f(\omega)$ over state X (coding function), and express the right signals received by a DC formula $g(\omega)$ over state Y (the inverse function of the decoding function).

The discrete property of state Y according to the receiver's clock can be specified by the DC* formula
$\ell<\Delta;discrete(\Delta,Y);\ell<\Delta,$

where $discrete(\Delta, Y) =_{\text{def}} ((\lceil Y \rceil \vee \lceil \neg Y \rceil) \wedge \ell = \Delta)^*$.

By arithmetic calculation we can prove easily that:

$$\bigwedge \left\{ \begin{array}{l} \ell = r; \ell \geq 2\Delta \\ \ell < \Delta; discrete(\Delta, Y); \ell < \Delta \end{array} \right\}$$
$$\Rightarrow \left(\begin{array}{l} r \leq \ell < r + \Delta; \\ discrete(\Delta, Y); \ell < \Delta \end{array} \right)$$

Therefore, we have that $(\mathcal{A}1) \quad X \rhd_\delta Y, \quad (\mathcal{A}2) \quad \neg X \rhd_\delta \neg Y.$

Let $\mathcal{L}_{DC}$ be the language of DC, and define the string functions $f : \{0,1\}^+ \rightarrow \mathcal{L}_{DC}$ and $g : \{0,1\}^+ \rightarrow \mathcal{L}_{DC}$ as follows. Let
$hhs(X) =_{\text{def}} \lceil X \rceil \wedge \ell = b; \lceil X \rceil \wedge \ell = a$
$hls(X) =_{\text{def}} \lceil X \rceil \wedge \ell = b; \lceil \neg X \rceil \wedge \ell = a$
$lhs(X) =_{\text{def}} \lceil \neg X \rceil \wedge \ell = b; \lceil X \rceil \wedge \ell = a$
$lls(X) =_{\text{def}} \lceil \neg X \rceil \wedge \ell = b; \lceil \neg X \rceil \wedge \ell = a$
$ends(State) =_{\text{def}} \lceil State \rceil \wedge \ell = b + a$
$hler(Y) =_{\text{def}} \lceil Y \rceil; (\lceil \neg Y \rceil; true) \wedge \ell = d\Delta$
$lher(Y) =_{\text{def}} \lceil \neg Y \rceil; (\lceil Y \rceil; true) \wedge \ell = d\Delta$

Then $f(0) =_{\text{def}} \lceil \neg X \rceil \wedge \ell = b + a; hhs(X); ends(X)$
$f(1) =_{\text{def}} \lceil \neg X \rceil \wedge \ell = b + a; hls(X); ends(\neg X)$

$$f(\omega 0) =_{\text{def}} \left\{ \begin{array}{ll} \phi; hhs(X); end(X) & \text{if } f(\omega) = \phi; ends(\neg X) \\ \phi; lls(X); end(\neg X) & \text{if } f(\omega) = \phi; ends(X) \end{array} \right.$$

$$f(\omega 1) =_{\text{def}} \left\{ \begin{array}{ll} \phi; hls(X); end(\neg X) & \text{if } f(\omega) = \phi; ends(\neg X) \\ \phi; lhs(X); end(X) & \text{if } f(\omega) = \phi; ends(X) \end{array} \right.$$

$g(0) =_{\text{def}} \ell < \delta; \lceil \neg Y \rceil \wedge \ell > b + a - \delta; (\lceil Y \rceil; true) \wedge \ell = d\Delta; \lceil Y \rceil$

$g(1) =_{\text{def}} \ell < \delta; \lceil \neg Y \rceil \wedge \ell > b + a - \delta; (\lceil Y \rceil; true) \wedge \ell = d\Delta; \lceil \neg Y \rceil$

$$g(\omega 0) =_{\text{def}} \left\{ \begin{array}{ll} \psi; lher(Y); \lceil Y \rceil & \text{if } g(\omega) = \psi; \lceil \neg Y \rceil \\ \psi; hler(Y); \lceil \neg Y \rceil & \text{if } g(\omega) = \psi; \lceil Y \rceil \end{array} \right.$$

$$g(\omega 1) =_{\text{def}} \left\{ \begin{array}{ll} \psi; lher(Y); \lceil \neg Y \rceil & \text{if } g(\omega) = \psi; \lceil \neg Y \rceil \\ \psi; hler(Y); \lceil Y \rceil & \text{if } g(\omega) = \psi; \lceil Y \rceil \end{array} \right.$$

For each $\omega \in \{0,1\}^+$, the DC formula $f(\omega)$ represents the coding of BMP and $g(\omega)$ represents the inversion of the decoding, i.e. $g(\omega)$ represents the state of received signals that results ω after decoding. So the protocol will be correct if for all $\omega \in \{0,1\}^+$, $f(\omega)$ implies $g(\omega)$ in DC.

Let ***As*** $=_{\text{def}} \{(\mathcal{A}1), (\mathcal{A}2)\}$. The correctness of the protocol is formulated as:

Theorem 2 *Assume that the conditions $(\mathcal{C}1)$ and $(\mathcal{C}2)$ hold. Then, the following hold:*

(1) *For all $\omega, \omega' \in \{0,1\}^+$ such that $\omega \neq \omega'$, we have:* $\vdash_{DC} g(\omega) \wedge g(\omega') \Rightarrow false$

(2) *For all $\omega \in \{0,1\}^+$, As* $\vdash_{DC} f(\omega) \Rightarrow g(\omega)$

The readers are referred to [4] for a formal proof of the theorem.

Based on the constraints $(\mathcal{C}1)$ and $(\mathcal{C}2)$ we can determine easily values of parameters in the design of BMP. For example, in case $b = 5$, $a = 13$ and $r = 1$, if we choose $d = 10$, the parameter θ must satisfy the inequalities:

$$\frac{d+2}{b+a-r} < \theta \leq \frac{d-1}{b+r}$$

Hence, the protocol is correct if the ratio of clock rates is within $min(1 - \frac{6}{9}, \frac{17}{12} - 1) = \frac{1}{3} \approx 33\%$ of unity.

If the ratio of clock rates is given to be within $\alpha = 10\%$, we have $\frac{1}{1+\alpha} \leq \theta \leq \frac{1}{1-\alpha}$. Hence, $8 \leq d \leq 13$

CONCLUSION & FUTURE WORK

We have proposed a systematic design of real-time systems using DC. The main idea is to model the system as a set of state variables that consists of two kinds of states: continuous states and discrete states. From the requirements and the interface between continuous and discrete states we derive the designs step by step and optimise them by using auxiliary state variables. Our method can also discover useful constraints on parameters which can ensure the correctness of the design in general.

We have also considered the biphase mark protocol and shown that our model of verification works well for this case. Comparing with methods in [3] and [5] our method has some advantages. It models the protocol in a more natural way with more detailed physical assumptions and higher accuracy. Moreover, it can help us to choose optimal values of the parameters in the design. By combining DC, formal languages together with the induction principle, we have developed a new technique for specifying the BMP and proving its correctness. The verification method presented in this paper can be used for other problems as well.

In our future work, we will develop a set of rules for the calculation of more detailed designs based on the model and techniques presented in this paper.

References

[1] Michael R. Hansen and Zhou Chaochen. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9: 283-330, 1997.

[2] Dang Van Hung. Modelling and Verification of Biphase Mark Protocols in Duration Calculus Using PVS/DC$^-$. Proceedings of the *1998 International Conference on Application of Concurrency to System Design* (CSD'98), 23-26 March 1998, Aizu-wakamatsu, Fukushima, Japan, IEEE Computer Society Press, 1998, pp. 88 - 98.

[3] Dang Van Hung and Ko Kwang Il. Verification via Digitized Models of Real-Time Hybrid Systems. In *Proceedings of Asia-pacific Software Engineering Conference (APSEC'96)*, pages 4-15. IEEE Computer Society Press, 1996.

[4] Do Van Nhon and Dang Van Hung. A Systematic Design of Real-time Systems Using Duration Calculus. Technical Report 197, UNU/IIST, P.O.Box 3058, Macau, May 2000.

[5] J Strother Moore. A Formal Model of Asynchronous Communication and its Use in Mechanically Verifying a Biphase Mark Protocol. *Formal Aspects of Computing*, 6: 60-91, 1994.

[6] Francois Siewe and Dang Van Hung. From Continuous Specification to Discrete design. Technical Report 182, UNU/IIST, P.O.Box 3058, Macau, 1999.

[7] Rajeev Alur and David L. Dill. A theory of Timed Automata. *Theoretical Computer Science*, 126: 183-235, 1994.

[8] Mathai Joseph. Real-time Systems: Specification, Verification and Analysis. *Prentice Hall International (UK)*, 1996.