Electronic Notes in Theoretical Computer Science 65 No. 5 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume65.html 15 pages

The Saga of Synchronous Bus Arbiter: On Model Checking Quantitative Timing Properties of Synchronous Programs

Paritosh K. Pandya [1,2]

School of Technology and Computer Science
Tata Institute of Fundamental Research
Homi Bhabha Road, Colaba,
Mumbai 400005, India

Abstract

Quantified Discrete-time Duration Calculus, ($QDDC$), is a form of interval temporal logic [14]. It is well suited to specify quantitative timing properties of synchronous systems. An automata theoretic decision procedure for $QDDC$ allows converting a $QDDC$ formula into a finite state automaton recognising precisely the models of the formula. The automaton can be used as a synchronous observer for model checking the property of a synchronous program. This theory has been implemented into a tool called DCVALID which permits model checking $QDDC$ properties of synchronous programs written in Esterel, Verilog and SMV notations.

In this paper, we consider two well-known synchronous bus arbiter circuits (programs) from the literature. We specify some complex quantitative properties of these arbiters, including their response time and loss time, using $QDDC$. We show how the tool DCVALID can be used to effectively model check these properties (with some surprising results).

1 Introduction

For synchronous programs, execution time is measured in terms of clock ticks, i.e. the notion of time is discrete. For many such programs, it is important to analyse quantitative timing properties such as response time and latency. Unfortunately, such analysis has not received adequate attention from the program verification community.

[1] Partially supported by the UNU/IIST offshore project *Semantics and verification of real-time programs using Duration Calculus: Theory and Practice*
[2] Email: pandya@tifr.res.in

Quantified Discrete-time Duration Calculus, ($QDDC$), is a logic well suited to specifying such quantitative timing properties. $QDDC$ is a form of interval temporal logic with primitives which *count* the number of occurrences of a signal in a given behaviour fragment. It is a highly expressive logic which can succinctly specify many complex properties. Thus, $QDDC$ addresses a qualitatively different class of properties of synchronous programs from those considered earlier [4,18,3,11].

In order to illustrate this, in this paper, we consider two well-known synchronous bus arbiter circuits (programs). Using $QDDC$, we specify some complex quantitative properties of these arbiters, including their response time and loss time.

In spite of its high expressive power $QDDC$ formulae can be model checked. An automata theoretic decision procedure allows converting a $QDDC$ formula into a finite state automaton recognising precisely the models of the formula [14]. The automaton can be used as a synchronous observer for model checking the property of a synchronous program [8,15]. We have implemented this theory into a tool called DCVALID which permits model checking $QDDC$ properties of synchronous programs written in Esterel, Verilog and SMV notations [13,16].

In the paper, we show how the tool DCVALID can be used to effectively model check the timing properties of the arbiter circuits, with some surprising results. It is our claim these properties are quite difficult to analyse by hand and a system designer's intuition about them can be misleading. Hence, the availability of tools is crucial for the verification such properties.

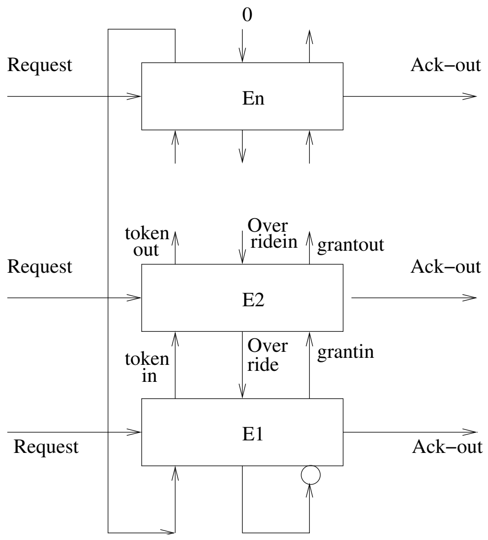
The rest of the paper is organised as follows. Some synchronous bus arbiter circuits are introduced in Section 1.1. A brief overview of logic $QDDC$ is given in Section 2. The automata theoretic approach to its model checking is also briefly outlined. Properties of the arbiters are formalised in Section 3. The analysis of these properties, carried out by our tool DCVALID, is presented in Section 4. The paper ends with a brief summary.

1.1 Synchronous Bus Arbiter

Example 1.1 A synchronous bus arbiter with n cells has request lines `req1`, …, reqi, …, reqn and acknowledgement lines `ack1`, …, acki, …, ackn. At any clock cycle a subset of the request lines are high. It is the task of the arbiter to set at most one of the corresponding acknowledgement lines high. Preferably, the arbiter should be fair to all requests. MacMillan [11] proposed the circuit of Figure 1 for the bus arbiter [3]. He also analysed its basic properties such as the mutual exclusion of `ack` signals using the pioneering SMV verifier. This circuit also can be encoded as an `Esterel module`. We present two variants of MacMillan's arbiter in Figure 4. (The changes from the original arbiter

[3] The circuit elements are standard. The square box denotes a *D-latch* which delays the signal by one clock cycle.

Cell Interconnection



Cell Circuit

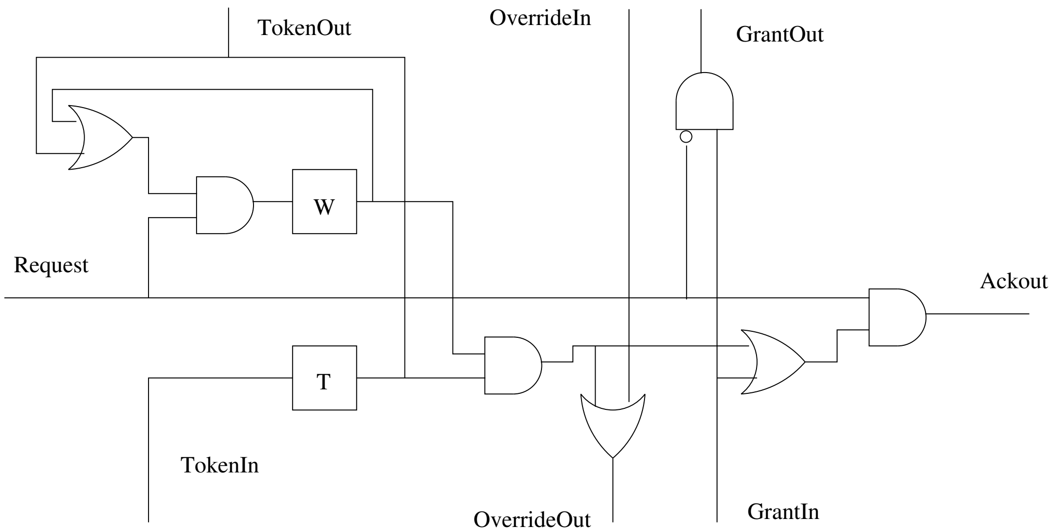
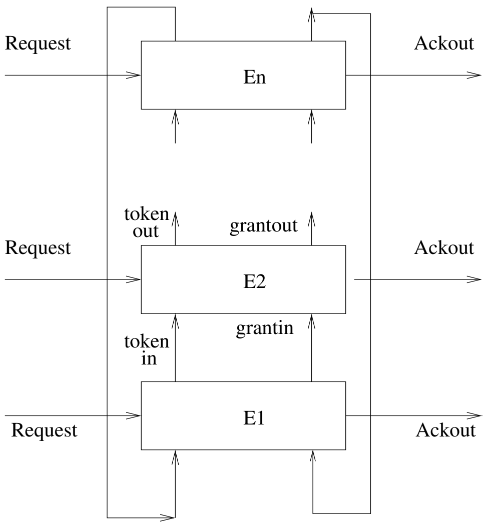


Fig. 1. MacMillan's Arbiter: Macarb

are highlighted by dotted lines.) The first variant *MacarbV1* arose due to a mistaken translation of the original arbiter code into Verilog and the second variant *MacarbV2* is due to Rahul Jain. A different arbiter circuit, shown in Figure 2, was proposed by de Simone [5] who also gave an `Esterel model` of this circuit. All these arbiters have the property that at most one `ack` signal can occur at a time. Existing model checkers such as Xeve [5], VIS [3] or SMV [11] can easily verify this invariance property. □

Example 1.2 We consider some complex properties of these arbiters which have not been investigated before. Many of these refer to quantitative timing aspects. Each of these properties must be analysed for a given arbiter circuit

Cell Interconnection



Cell Circuit

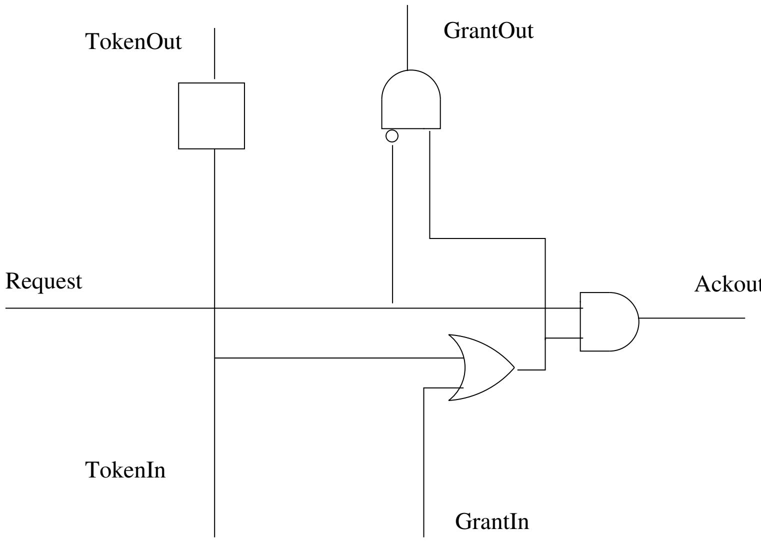


Fig. 2. de Simone's Arbiter: Simarb

of n cells for a given constant n.

(i) *Response time for cell i*: Is the worst case response time of cell i of the arbiter within m cycles? In other words, given a constant m is it the case that during any behaviour and in any time interval spanning m cycles if `reqi` is continuously high, there must be `acki`? Find the least such constant m.

(ii) *3-cycle response time for cell i*: Is the worst case response time for getting 3 `acki` signals within k cycles? Find the least such constant k.

(iii) A cycle is *lost* if at least one of the cells has its `req` high but all the cells have `ack` low. Can the arbiter loose no more than l consecutive cycles? Find the least such l.

(iv) *Fifo(i,j) property*: If the `req` for cell i comes before that of cell j (and persists) will the `ack` for cell i definitely occur before that of cell j? In this case we say that Fifo(i,j) holds. Determine all pairs (i, j) with Fifo(i,j) property for a given arbiter circuit. $\square$

We urge the reader to try to intuitively answer the above questions for the four synchronous arbiter circuits presented in Example 1.1.

2 Quantified Discrete-time Duration Calculus

Quantified Discrete-time Duration Calculus, ($QDDC$), is an interval temporal logic for specifying properties of finite sequences of states (valuations). We give a brief overview of this logic.

Let $Pvar$ be a finite set of propositional variables representing some observable aspects of system state. Let $VAL(Pvar) \stackrel{\text{def}}{=} Pvar \rightarrow \{0, 1\}$ be the set of valuations assigning truth-value to each variable. We shall identify behaviours with finite, nonempty sequences of valuations, i.e. $VAL(Pvar)^+$.

Example 2.1 The following picture gives a behaviour over variables $\{p, q\}$. Each column vector gives a valuation, and the word is a sequence of such column vectors.

```
p   1  0  1  1  0
q   0  0  0  0  1
```

The above word satisfies the property that p holds initially and q holds at the end but nowhere before that. QDDC is a logic for formalising such properties. Each formula specifies a set of such words. $\square$

Given a non-empty finite sequence of valuations $\sigma \in VAL^+$, we denote the satisfaction of a QDDC formula D over σ by

$$\sigma \models D$$

We now give the syntax and semantics of QDDC and define the above satisfaction relation.

Syntax of QDDC Formulae

Let $Pvar$ be the set of propositional variables. Let p, q range over propositional variables, P, Q over propositions (boolean combinations of p, q) and D, D_1, D_2 over QDDC formulae. Let c range over natural number constants. The syntax of QDDC is as follows.

```
        <P> | [P] | [[P]] | D1^D2 | D1 && D2 | !D |
        ex P. D | slen op c | scount P op c
where
        op is in { <=, <, =, >, >= }
```

Let $\sigma \in VAL(Pvar)^+$ be a behaviour. Let $\#\sigma$ denote the length of σ and $\sigma[i]$ the i'th element. For example, if $\sigma = \langle v_0, v_1, v_2 \rangle$ then $\#\sigma = 3$ and $\sigma[1] = v_1$. Let $dom(\sigma) = \{0, 1, \ldots, \#\sigma - 1\}$ denote the set of positions within σ.

Let $\sigma, i \models P$ denote that proposition P evaluates to true at position i in σ. We omit this obvious definition.

The set of intervals in σ is $Intv(\sigma) = \{[b,e] \in dom(\sigma)^2 \mid b \leq e\}$ where each interval $[b,e]$ identifies a subsequence of σ between positions b and e.

We inductively define the satisfaction of a QDDC formula D for behaviour σ and interval $[b,e] \in Intv(\sigma)$ as follows. This is denoted by $\sigma, [b,e] \models D$.

$$\sigma, [b,e] \models \texttt{<P>} \quad \textbf{iff} \quad b = e \text{ and } \sigma, b \models \texttt{P}$$

$$\sigma, [b,e] \models \texttt{[[P]]} \quad \textbf{iff} \quad \sigma, i \models \texttt{P} \text{ for all } i : b \leq i \leq e$$

$$\sigma, [b,e] \models \texttt{[P]} \quad \textbf{iff} \quad b < e \text{ and } \sigma, i \models \texttt{P} \text{ for all } i : b \leq i < e$$

$$\sigma, [b,e] \models \texttt{!D} \quad \textbf{iff} \quad \sigma, [b,e] \not\models D$$

$$\sigma, [b,e] \models \texttt{D1 \&\& D2} \quad \textbf{iff} \quad \sigma, [b,e] \models \texttt{D1} \text{ and } \sigma, [b,e] \models \texttt{D2}$$

$$\sigma, [b,e] \models \texttt{D1 \^{} D2} \quad \textbf{iff} \quad \text{for some } m : b \leq m \leq e :$$
$$\sigma, [b,m] \models \texttt{D1} \text{ and } \sigma, [m,e] \models \texttt{D2}$$

Entities `slen` and `scount P` are called *measurements*. Term `slen` denotes the length (in number of steps) of the interval whereas `scount P` denotes the count of number of times P is true within the interval $[b,e]$ Formally,

$$eval(\texttt{slen}, \sigma, [b,e]) \stackrel{\text{def}}{=} e - b$$

$$eval(\texttt{scount P}, \sigma, [b,e]) \stackrel{\text{def}}{=} \sum_{i=b}^{e} \left\{ \begin{array}{ll} 1 & if \quad \sigma, i \models \texttt{P} \\ 0 & otherwise \end{array} \right\}$$

Let t range over measurements. Then,

$$\sigma, [b,e] \models t \ op \ c \quad \textbf{iff} \quad eval(t, \sigma, [b,e]) \ op \ c$$

Note that `slen=3` holds for a state sequence with 3 steps, i.e. 4 states. Also `scount P = 3` holds for a sequence where `P` is true 3 times.

Call a behaviour σ' to be p-variant of σ provided $\#\sigma = \#\sigma'$ and for all $i \in dom(\sigma)$ and for all $q \neq p$, we have $\sigma(i)(q) = \sigma'(i)(q)$. Then,

$$\sigma, [b,e] \models \texttt{ex p. D} \quad \textbf{iff} \quad \sigma', [b,e] \models \texttt{D} \text{ for some } p\text{-variant } \sigma' \text{ of } \sigma$$

Finally, $\sigma \models \texttt{D}$ **iff** $\sigma, [0, \#\sigma - 1] \models \texttt{D}$

We can also define some derived constructs. Boolean combinators `||, =>, <=>` denoting or, implies and equivalence can be defined using `&&, !` as usual.

- $\texttt{<>D} \stackrel{\text{def}}{=}$ `true^D^true` holds provided `D` holds for some subinterval.
- `[]D` $\stackrel{\text{def}}{=}$ `!<>!D` holds provided `D` holds for all subintervals.

Example 2.2 We give some examples of $QDDC$ formulae.

- The property of Example 2.1 can be formulated in $QDDC$ as follows. It specifies behaviours where P holds initially and Q holds at the end but nowhere before that.

```
<P>^[!Q]^<Q>
```

- The following formula holds for a behaviour σ provided for all fragments σ' of σ which have (a) P true in the beginning, (b) Q true at the end, and (c) no occurrences of Q in between, the number of occurrences of states in σ' where R is true is at most 3.

```
[] (<P>^[!Q]^<Q> => (scount R <= 3))
```

□

QDDC can specify safety and bounded-liveness properties of systems. Since it only specifies finite sequence of states, it cannot deal with general liveness properties. There are many extensions addressing this problem [15].

2.1 Model Checking QDDC

An execution α of a synchronous program P is a finite or infinite sequence of valuations (states). A $QDDC$ formula D holds (is valid) for an execution if *all* finite prefixes of α satisfy D. Finally, the formula is valid for program P if D is valid for all executions of D. Let $\alpha[i]$ denote the ith element of sequence α and let $\alpha[i:j]$ denote the subsequence between positions i to j (inclusive).

Definition 2.3 [Prefix Validity] Let $\alpha \models D$ **iff** $\alpha[0:e] \models D$ for all $e \in dom(\alpha)$. Let $P \models D$ **iff** $\alpha \models D$ for all executions α of P

The model checking problem is to determine by an algorithm whether $P \models D$. We outline an automata theoretic approach to model checking $QDDC$ below.

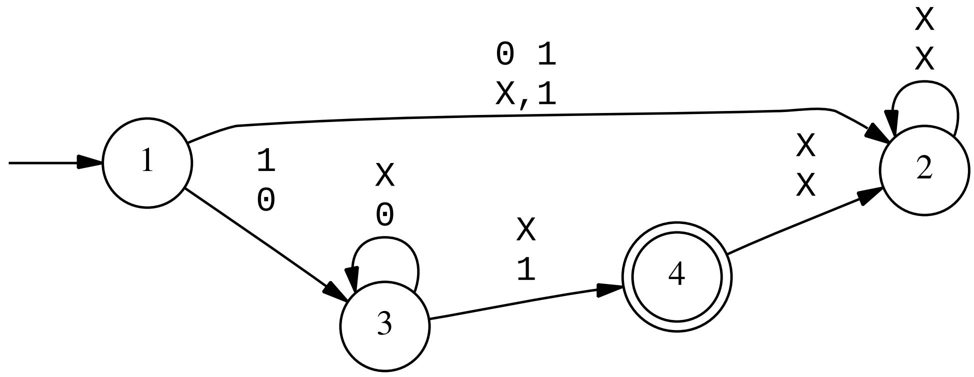
The following theorem characterises the sets of models of a QDDC formula. Let $pvar(D)$ be the finite set of propositional variables occurring within a QDDC formula D. Let $VAL(Pvar) = Pvar \rightarrow \{0,1\}$ be the set of valuations over $Pvar$.

Theorem 2.4 *For every QDDC formula D, we can effectively construct a finite state automaton $A(D)$ over the alphabet $VAL(pvar(D))$ such that for all $\sigma \in VAL(pvar(D))^*$,*

$$\sigma \models D \quad \textbf{iff} \quad \sigma \in L(A(D))$$

We refer the reader to [14] for a proof of this theorem. This construction has been implemented into a tool called DCVALID [13,14].

Example 2.5 The first property of Example 2.2 can be stated in QDDC as the formula <P>^[!Q]^<Q>. The automaton corresponding this formula is given below. Each edge is labelled with a column vector giving truth values of variables P, Q as in Example 2.1. Also, letter X is used to denote either 0 or 1. Note that the automaton is minimal, deterministic and total.



From this automaton, we can see that a model and a counter model of least length for the formula are as follows. (Empty words are not considered models in QDDC.)

```
   model              counter-model
P  1 X              P 1
Q  0 1              Q 0
```

Given a *QDDC* formula D, we can use the automaton $A(\texttt{!D})$, obtained as in Theorem 2.4, as a *synchronous observer* for determining the violation of D during an execution. The basic idea is that the system M and the observer $A(\texttt{!D})$ are executed in synchronous parallel (lock-step) mode. This means we consider the execution of the transformed system

$$M' \quad \stackrel{\text{def}}{=} \quad M \parallel A(\texttt{!D})$$

The observer is a total, deterministic automaton. It does not affect or constrain the activity of M in any way. At any point in an execution of M', the observer will have observed the state sequence arising in the M component. If this state sequence satisfies !D then the observer will be in its final state, otherwise the observer will be in its non-final state. Thus, M violates D if and only if there is some execution of M' during which the observer for !D enters its final state. (See Halbwachs *et al* [8] for details of the observer approach to verification.)

Theorem 2.6 $M \models D$ **iff**
A final state of $A(\texttt{!D})$ *cannot be reached in* $(M \parallel A(\texttt{!D}))$

We omit the proof of this theorem which can be found elsewhere [15].

If M is a finite state system then so is $(M \parallel A(\texttt{!D}))$. For such systems, the reachability of final states can be analysed by symbolic breadth-first search [11]. There are now many mature model checking tools which can perform this search quite efficiently. For example, if M and $A(\texttt{!D})$ are given as Esterel modules, the Esterel verification tool Xeve [5] can perform this search. Similarly, if they are given as SMV modules then the SMV tool [11] can perform the search and if they are given as Verilog modules the VIS tool [3] can perform the search. Note that the modelling languages of all these tools support synchronous parallel composition permitting the observer to be run synchronously with the system. Exploiting these search procedures, we have constructed a model checking tool for checking *QDDC* properties of Esterel, SMV and Verilog designs.

2.2 Tool DCVALID

The reduction from formulae of *QDDC* to finite state automata, as outlined in Theorem 2.4, has been implemented into a tool called DCVALID [13,14]. The tool generates a total, deterministic and minimal automaton for a formula and it can also checks for the validity of the formula by searching for rejecting paths in the automaton. The automaton in Example 2.5 was automatically generated from the formula by the tool DCVALID.

An associated tool, called CTLDC, translates the automaton into Esterel, SMV or Verilog module to give a synchronous observer for the property. It also connects the module to run synchronously with a given system module. The resulting program can be analysed for reachability of accepting/rejecting states using existing tools such as Xeve [5], SMV [11] and VIS [3]. This determines whether a QDDC property is valid for the given system module as stated in Theorem 2.6. Thus, DCVALID can determine whether $M \models D$ where M a (pure) Esterel, SMV or Verilog program and D is a *QDDC* formula. If the verification fails the tool generates a counter-example.

The details of logic *QDDC*, the architecture of tool DCVALID and some performance statistics are described elsewhere [14,16] and omitted here for brevity. Basically, DCVALID is built on top of MONA [7]. MONA is an efficient and sophisticated implementation of the automata-theoretic decision procedure of Buchi and Elgot for monadic logic over finite words (MLSTR). DCVALID works by translating a QDDC formula into a boolean combination of MLSTR formulae. Each component is then translated into an automaton by MONA. Although the worst-case complexity is non-elementary the tool is often able to handle reasonably large formulae [14].

3 Specification of Synchronous Bus Arbiter in *QDDC*

We now formalise in *QDDC* the four properties of a synchronous bus arbiter which where given earlier in Example 1.2.

(i) *Response Time for cell i* Is the worst case response time of cell i of an arbiter less than or equal to m cycles? Formally, is the following formula valid for a given arbiter circuit?
`[]  ( [[Reqi]] && (slen = m-1) => <><Acki> )`
Find the minimum m which makes this formula valid for a given arbiter.

(ii) *3-cycle Response Time for cell i* Is the worst case response time for obtaining 3 `ack`i signals less than or equal to k cycles? Formally is the following formula valid for the arbiter?
`[]  ( [[Reqi]] && (slen = k-1) => (scount Acki >= 3))`

Find the minimum k which makes this formula valid for a given arbiter.

(iii) *Loss-time* Can the arbiter loose no more than l consecutive cycles? Formally, is the following formula valid for the arbiter?
`[] ([[lostcycle]] => slen <= l-1 )`
Find the minimum l which makes the formula valid for a given arbiter.

(iv) *FIFO(i,j) property* Find (i,j) pairs such that the following formula is valid for the given arbiter. This formula states that if request for cell i arrives before that of cell j and it persists, then the acknowledgement for cell j must not occur before that of cell i.
`[]( (<!reqj>^[[reqi && !acki]]) => !<><ackj>)`

4 Model Checking of Arbiter Properties

A model checker for $QDDC$ was described in Section 2.2. For a given n cell arbiter with fixed constant n, the DCVALID tool (together with a reachability analysis tool such as Xeve), can check whether a formula such as

```
[]  ( [[Reqi]] && (slen = k-1) => (scount Acki >= 3))
```

is valid for the arbiter *for a given value of constant k*. In order to find the minimum k making this formula valid, we must try different values of k.

The four properties of synchronous bus arbiters given in Section 3 were checked for various 5-cell arbiter circuits. The properties were verified for de Simone's arbiter (*Simarb*), MacMillan's arbiter (*Macarb*) as well as two variants of MacMillan's arbiter, respectively called *MacarbV1* and *MacarbV2*. The circuit diagrams of these arbiters have been given earlier in Example 1.1.

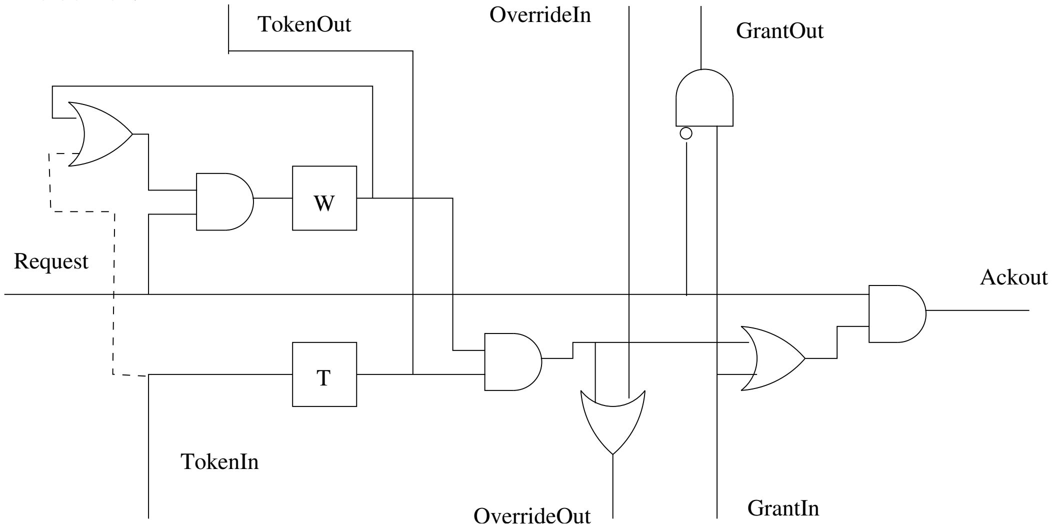
The verification experiments were carried out using the tools DCVALID and Xeve/SMV and the properties were checked against both Esterel and SMV code for the circuits. de Simone's arbiter was coded only in Esterel as it is not possible to code this circuit directly into notations such as Verilog or SMV. This circuit contains a potential combinational cycle and a sophisticated causality analysis [2] of Esterel is needed to handle it.

We present the results of model checking below. In each case the *minimum constant* making the formula valid is given. This was found by trial and error. (The reader may refer to Section 3 for the definitions of the verified properties.)

Property 1: Response time m for cell i

Simarb: $m = 5$ cycles for cells $i = 1$ to 5
Macarb: $m = 5$ cycles for cell $i = 1$
Macarb: $m = 10$ cycles for cells $i = 2$ to 5
MacarbV1: $m = 5$ cycles for cell $i = 1$
MacarbV1: $m = 6$ cycles for cells $i = 2$ to 5
MacarbV2: $m = 5$ cycles for cell $i = 1$
MacarbV2: $m = 10$ cycles for cells $i = 2$ to 5

MacarbV1



MacarbV2

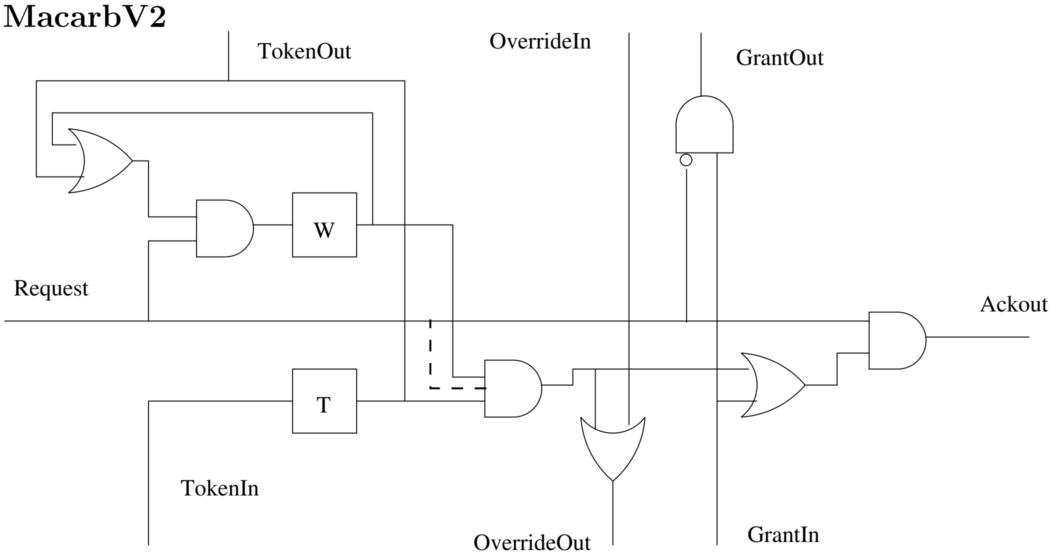


Fig. 3. Variants of MacMillan's Arbiter

Property 2: 3-cycle Response time k for cell i

Simarb: $k = 15$ cycles for cells $i = 1$ to 5
Macarb: $k = 15$ cycles for cell $i = 1$
Macarb: $k = 20$ cycles for cells $i = 2$ to 5
MacarbV1: $k = 15$ cycles for cell $i = 1$
MacarbV1: $k = 16$ cycles for cells $i = 2$ to 5
MacarbV2: $k = 15$ cycles for cell $i = 1$
MacarbV2: $k = 20$ cycles for cells $i = 2$ to 5

Property 3: Maximum Number l of Consecutive Lost Cycles

Simarb: $l = 0$, i.e. no lost cycles.
Macarb: $l = 5$, i.e. at most 5-cycles lost in a row.
MacarbV2: $l = 0$, i.e. no lost cycles.
In case of *MacarbV1* for all large values of l tried the property fails or the checker runs out of time/memory. Thus, no upper bound on l could be found.

Property 4: Fifo(i,j) Pairs

We list exactly those (i,j) pairs for which Fifo(i,j) property holds.
Simarb: (1,2), (2,3), (3,4), (4,5), (5,1)
Macarb: (1,2), (1,3), (1,4), (1,5), (2,3), (3,4), (4,5)
MacarbV1: (1,2), (1,3), (2,3), (3,4), (4,5)
MacarbV2: (1,2), (1,3), (1,4), (1,5), (2,3), (3,4), (4,5)

A significant aspect of this model checking is that if the verification concludes that the property is not valid then a counter example scenario is generated. For example, if *Property 3* is checked for MacMillan's 5-cell arbiter with value of $l = 4$, the tool reports that the property is not valid and gives an execution of the arbiter which violates the property. Such a counter example, generated using DCVALID and Xeve, is given below. If we simulate the arbiter on this input we find that in fact more than 4 cycles are lost consecutively.

```
req1 ;
req1 req2 ;
req1 req2 req3 ;
req1 req2 req3 req4 ;
req1 req2 req3 req4 req5 ;
req2 req3 req4 req5 ;
req3 req4 req5 ;
req4 req5 ;
req5 ;
req4 ;
```

Performance

QDDC is a highly expressive logic which can succintly specify complex properties. In the worst case, the complexity of the automaton $A(D)$ can be non-elementary in the size of D, although this is rarely observed in practice. (See [14] for some statistics.) Here, we give raw performance figures for the verification of *Property 3* for MacMillan's arbiter, *Macarb*. For an n-cell arbiter the property was verified with $m = 2n$ for the cell $i = 3$. The tests were carried out on a 1.4GHz Pentium 4 processor machine with 512 Mbytes of Ram running Linux 2.4.16 kernel. Esterel V5.92 and SMV Version 2.5.4.3 were used in the tests. All the time values are in seconds.

Cells	SMV		Esterel		
n	Observer Generation	Verification	Observer Generation	Compilation	Verification (checkblif)
5	0.06	0.01	0.06	0.15	1.27
10	0.13	0.07	0.07	0.28	35.21
100	1.06	3.51	1.06	3.09	**
200	4.67	140	–	–	–

** Esterel verification for 100-cell arbiter was aborted after 8 hours due to memory overrun.

5 Summary

QDDC is a highly expressive logic which allows complex properties of synchronous programs to be conveniently specified. It is especially suited to specification of quantitative timing properties such as the response time. Moreover, *QDDC* is supported by a model checking tool DCVALID which can effectively analyse these properties. *QDDC* is a discrete-time version of the dense time logic Duration Calculus [19,9]. The logic is expressive enough to allow us to write a compositional semantics of a synchronous language like pure Esterel in it [17].

We have shown that circuits/synchronous programs such as the synchronous bus arbiters of Example 1.1 embody a fair degree of complexity in their behaviour. The surprising values of the *response* and the *loss* times for MacMillan's Arbiter and its two variants should convince the reader that these properties are quite hard to analyse by hand. Similarly, the values of *(i,j)* satisfying the *Fifo(i,j)* property for various arbiters are also strange. Hence, logics and model checkers for timing properties of synchronous programs deserve serious investigation. Logic *QDDC* and tool DCVALID allow such analysis to be performed.

QDDC is a discrete-time interval temporal logic for specifying timing properties of synchronous programs. Other such logics include the RTCTL [6] of Emerson *et al* and the Synchronous Regular Timing Diagrams [1] of Amla *et al.* A tool set called TEMPEST has been developed by Jagadeesan *et al* for verifying safety and some response properties of Esterel programs [10].

It should be noted that all the timing properties verified in this paper have the form that $M \models D(l)$ holds for a given constant (natural number) l. For example, *Property 3* states that no more than l consecutive cycles can be lost. The real question is to find the least value of l for which the property holds. Methods presented in this paper allow such least/greatest values to be found only by trying out different values of l. This can be cumbersome and sometimes impossible. For example, in verifying *Property 3* for the arbiter *MacarbV1*, we found that the property does not hold for any possible value of constant l tried. However, is it possible that the property does hold for some very large untried value of l. Recently, we have extended our tool DC-VALID with methods which can "compute" the least/greatest values of these constants. Using these methods we have determined that *Property 3*, in fact, does not hold for *any* value of constant l and the arbiter *MacarbV1* can loose unboundedly many consecutive cycles. A separate paper, in preparation, will present these techniques and results.

References

[1] Amla, N., E. Allen Emerson, R.P. Kurshan and Kedar S. Namjoshi, Model checking synchronous timing diagrams, in *Proc. FMCAD*, LNCS 1954, Springer-Verlag, 2000.

[2] Berry, G., *The constructive semantics of Esterel*, 1999.

[3] Bryton, R.K., G.D. Hatchtel *et al*, VIS: A system for verification and synthesis, in *Proc. Computer Aided Verification, CAV'96*, Lecture Notes in Computer Science 1102, Springer-Verlag, 1996.

[4] Bouali, A., J.P. Marmorat, R. de Simone and H. Toma, Verifying Synchronous Reactive Systems Programmed in Esterel, in *Proc. of FTRTFT'96*, LNCS, Springer-Verlag, 1996.

[5] Bouali, A., XEVE: An Esterel Verification Environment, *Proc. Computer Aided Verification, CAV'98*, Lecture Notes in Computer Science, Springer-Verlag, 1998.

[6] Emerson, E.A., A.K. Mok, A.P. Sistla and J. Srinivasan, Quantitative temporal reasoning, In *Proc. CAV'90*, LNCS, Springer-Verlag, 1990.

[7] Henriksen, J.G., J. Jensen, M. Jorgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm, Mona: Monadic Second-Order Logic in Practice, in *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95*, LNCS 1019, 1996.

[8] Halbwachs, N., F. Lagnier and P. Raymond, Synchronous observers and the verification of reactive systems, in *Proc. Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, Twente, Springer-Verlag, 1993.

[9] Hansen, M.R. and Zhou Chaochen, Duration Calculus: Logical Foundations, *Journal of Formal Aspects of Computing* **9**, 1997.

[10] Jagadeesan, L.J., C. Pouchol and J.E. Von Olnhausen, Safety property verification of ESTEREL programs and appliactions to telecommunications software. In *Proc. CAV'95*, LNCS 939, Springer-Verlag, 1995.

[11] McMillan, K., *Symbolic Model Checking*, Kluwer Academic Publisher, 1993.

[12] Moszkowski, B., A Temporal Logic for Multi-Level Reasoning about Hardware, in *IEEE Computer*, **18**(2), 1985.

[13] Pandya, P.K., DCVALID User Manual, Tata Institute of Fundamental Research, Bombay, 1997. (Available in revised version at `http://www.tcs.tifr.res.in/~pandya/dcvalid.html`).

[14] Pandya, P.K., Specifying and Deciding Quantified Discrete-time Duration Calculus Formulae using DCVALID: An Automata Theoretic Approach, In *Proc. Workshop on Real-time Tools (RTTOOLS'2001)*, Aalborg, Denmark, August 2001.

[15] Pandya, P.K., Model checking CTL*[DC], In *Proc. TACAS 2001*, Genova, Italy, LNCS 2031, Springer-Verlag, 2001.

[16] Pandya, P.K., Model checking CTL[DC] specifications of SMV, Verilog and Esterel Designs, Technical Report, TCS-00-PKP-3, Tata Institute of Fundamental Research, September 2000.

[17] Pandya, P.K., Y.S. Ramakrishna and R.K. Shyamasundar. A Compositional Semantics of Esterel in Duration Calculus. In *Proc. Second AMAST workshop on Real-time Systems: Models and Proofs*, Bordeux, June, 1995.

[18] Raymond, P., Recognizing Regular Expressions by means of Dataflows Networks, In *Proc. 23rd International Colloquium on Automata, Languages, and Programming, (ICALP'96)*, LNCS 1099, Springer Verlag. Paderborn (Germany), July 1996.

[19] Zhou Chaochen, C.A.R. Hoare and A.P. Ravn, A Calculus of Durations, *Info. Proc. Letters*, **40**(5), 1991.