Conforming the Runtime Inputs for Hard Real-Time Embedded Systems

Kai Huang
fortiss GmbH, Germany
khuang@fortiss.org

Gang Chen
Technical University Munich, Germany
gangchen1170@tum.edu

Christian Buckl
fortiss GmbH, Germany
buckl@fortiss.org

Alois Knoll
Technical University Munich, Germany
knoll@in.tum.de

ABSTRACT

Timing is an important concern when designing an embedded system. While lots of researches on hard real-time systems focus on design-time analysis, monitoring the corresponding runtime behaviors are seldom investigated. In this paper, we investigate the conformity problem for runtime inputs of a hard real-time system. We adopt the widely used arrival curve model which captures the worst/best-cases event arrivals in the time interval domain and propose an algorithm to on-the-fly evaluate the conformity of the system input w.r.t. given arrival curves. The developed algorithm is lightweight in terms of both computation and memory overheads, which is particularly suitable for resource-constrained embedded systems. We also provide proofs and an FPGA implementation to demonstrate the effectiveness of our approach.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Real-time and embedded systems; B.8.0 [**Hardware**]: Performance and Reliability—*General*

General Terms

Algorithms, Design, Performance

Keywords

Real-Time Calculus, Leaky Bucket, Greedy Shaper

1. INTRODUCTION

Guaranteeing timing properties is an important aspect for building embedded systems. In particular for the class of real-time embedded systems, meeting timing constraints, e.g., worst-case response time and end-to-end latencies, is a major design concern. Researchers on hard real-time timing analysis in the literature [1, 5, 6, 12] often focus on design-time analysis, trying to compute worst-case bounds on timing properties at an early phase of the system design. The validity of the design-time analysis and the safeness of the derived bounds rely on the assumption that the system input follows certain specifications. In order to not harm the safeness of the analysis results, the runtime inputs of the system (or components) need to be conformed to the specifications used by the design-time analysis.

The conformity verification, however, is non-trivial. On the one hand, the verification has to cover the worst cases in order to be in consistence with the offline analysis. On the other hand, the verification and a possibly preceeding regulation mechanism have to be lightweight due to the stringent timing and resource budgets of the system. Therefore, directly applying the commonly used techniques which usually rely on expensive numerical computation may not be suitable for the runtime monitoring.

In this paper, we investigate the runtime conformity problem. Targeting hard real-time embedded systems, we try to provide on-the-fly verification for the worst-case conformity of system inputs. We adopt the widely used arrival-curve model which captures the worst/best-cased system inputs in the time interval domain and propose an algorithm to evaluate the conformity of input traffic with respect to given arrival curves. In case too many events are detected, our algorithm regulates the traffic such that the traffic complies again with the curve specifications assumed at design time. In case too few events are detected, no generic solution can be offered, but the applications can be notified.

Based on the results in [9] that an arrival curve can be conservatively approximated by a set of staircase functions each of which can be modeled by a leaky bucket, we use a dual-bucket mechanism to monitor each staircase function during runtime, one for conformity verification and one for traffic regulation. By tracking the fill level of buckets, the computationally expensive (de-)convolutions used by the design-time analysis are eliminated. Our approach is thus lightweight in terms of both computational overhead and memory footprint, particularly suitable for embedded systems with limited resources. We also provide formal proofs and an FPGA prototype for our algorithm to demonstrate the effectiveness of our approach.

The rest of this paper is organized as follows: Section 2 reviews related work in the literature. Section 3 presents the system models and analyzes the problem. Section 4 presents our algorithm and the proofs. Experimental results are presented in Section 5 and Section 6 concludes the paper.

2. RELATED WORK

The analysis of traffic regulators is not new. In the domain of classical networking flow control, the studies of lossless greedy regulators by means of network calculus can be found in [2, 11]. Such traffic regulators are usually modeled as leaky-bucket shapers. To model lossy systems, traffic clippers [4, 3] are introduced to regulate network packets. Unlike shapers that delay network packets, a traffic clipper actively discards non-conformed packets. The modeling of leaky-bucket greedy shapers in the context of real-time calculus (RTC) [12] is presented in [13]. The latest work on this direction [7] uses a greedy shaper to optimally reduce the peak temperature of a real-time system. All aforementioned work is offline analysis. Whether such analysis can be applied for online monitoring is not clear.

In [9], a methodology for coupling timed automata-based [1] and RTC-based models are proposed, which enables hybrid analysis of a system containing both state-based (timed automata) and state-less (RTC) components. The basic idea underlying the proposed methodology is to use a set of leaky-bucket event generators as an interface between the state-based and stateless abstractions, such that models can be interchanged between these two abstractions. This technique is applied in [8] to predict tighter worst-case bounds for the arrivals of future workload. Taking the idea in [9, 8] to the level of runtime, this paper investigates traffic conformance, trying to develop lightweight routines for on-the-fly traffic verification and regulation. We also prototyped an FPGA implementation of our approach on an ALTERA Cyclone III development board.

3. MODELS AND PROBLEM

Event Arrival Curves.

Event streams in a system can be described using a cumulative function $R(s, t)$, defined as the number of events seen in the time interval $[s, t)$. While any R always describes *one* concrete trace, a 2-tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ of upper and lower *arrival curves* [10] provides an abstract event stream model that characterizes a whole class of (non-deterministic) event streams. $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ provide an upper and a lower bound on the number of events seen on an event stream in *any* time interval of length Δ:

$$\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s),\ \forall\, 0 \leq s \leq t, \quad (1)$$

with $\alpha^l(\Delta) = \alpha^u(\Delta) = 0$ for $\Delta \leq 0$.

The concept of arrival curves unifies many other common timing models of event streams. For example, a periodic event stream can be modeled by a set of step functions where $\alpha^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$ and $\alpha^l(\Delta) = \lfloor \frac{\Delta}{p} \rfloor$. For a sporadic event stream with minimal inter arrival distance p and maximal inter arrival distance p', the upper and lower arrival curve is $\alpha^u(\Delta) = \lfloor \frac{\Delta}{p} \rfloor + 1$, $\alpha^l(\Delta) = \lfloor \frac{\Delta}{p'} \rfloor$, respectively. A widely used model to specify an arrival curve is the PJD model by which an arrival curve is characterized with a period p, jitter j, and minimal inter arrival distance d. The upper arrival curve is thus $\alpha^u(\Delta) = \min\{\lceil \frac{\Delta+j}{p} \rceil, \lceil \frac{\Delta}{d} \rceil\}$. For details, please refer to [12].

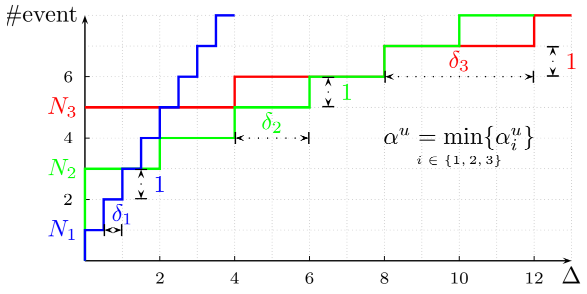


Figure 1: An example for an arrival curve as the combination of staircase functions.

Complex Arrival Patterns.

In this work, we deal with discrete numbers of event arrivals and their arrival patterns. In principle any (discrete) complex arrival pattern can be bounded by a set of upper and lower staircase functions, as long as the system under consideration is monotone and time-invariant [9]. The *monotone* property means that a higher number of input events seen in an interval yields a higher number of output events in intervals of equal or larger sizes. The *time-invariant* property means that the system behavior depends on the system states only. No matter when this state is reached, the possible set of the reactions of the systems is always the same, independently upon the concrete time when the actual state is reached.

An upper arrival curve thereby can be conservatively approximated as the minimum on the set of staircase functions of the form $\alpha_i(\Delta) = N^u_i + \lfloor \frac{\Delta}{\delta^u_i} \rfloor$:

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \alpha^u(\Delta) \leq \min_{i \in n}(\alpha^u_i(\Delta)). \quad (2)$$

An example for such approximation is depicted in Fig. 1, where α^u is given as the minimum of three staircase functions $\alpha^u_1 = 1 + \lfloor \frac{\Delta}{0.5} \rfloor$, $\alpha^u_2 = 3 + \lfloor \frac{\Delta}{2} \rfloor$, and $\alpha^u_3 = 5 + \lfloor \frac{\Delta}{4} \rfloor$.

Analogously, a set of staircase functions and maximum of which, i.e., $\alpha^l(\Delta) \geq \max_{j \in m}(0, \alpha^l_j(\Delta))$, can be employed for approximating a lower curve, where $\alpha^l_j(\Delta) = -N^l_j + \lfloor \frac{\Delta}{\delta^l_j} \rfloor$.

Problem Statement.

Given a trace R, checking its conformity w.r.t to an arrival curve is theoretically not a problem. One can simply inspect the trace by the definition in Eqn. (1). In the case that $\exists\, s, t,\ 0 \leq s < t,\ R(t) - R(s) > \alpha^u(t-s)$ or $R(t) - R(s) < \alpha^l(t-s)$, a violation occurs. Alternatively, one can apply the min-plus de-convolution:

$$\sup_{u \geq 0} \Big\{ R(t+u) - R(u) \Big\} \stackrel{\text{def}}{=} R(t) \oslash R(t) > \alpha^u(t) \quad (3)$$

according to [10],

Once a violation is detected, the traffic can be regulated to re-conform again to the specified arrival curves, e.g., by imposing a certain delay for the over-bursty input events. A usual way is to use a greedy shaper σ such that[1]

$$(R \oslash R) \otimes \sigma \leq \alpha^u \quad (4)$$

The shaper σ can be simply the convex hull of α^u.

One might notice that above approaches require numerical computation for the min-plus (de-)convolution, which demands intensive computing power as well as large

[1]min-plus convolution: $f \otimes g \stackrel{\text{def}}{=} \inf_{0 \leq s < t} \Big\{ f(t-s) + g(s) \Big\}$

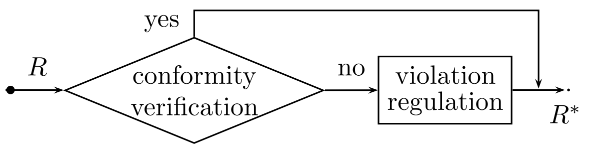


Figure 2: The flow of the approach.

memory footprint. Directly applying these approaches for online monitoring is thus prohibited, in particular for the class of embedded systems with stringent resource constraints. Therefore, lightweight alternatives are needed to conduct efficient conformance verification as well as violation recovery. In the next section, we will present an approach that solves this problem in a particular way.

4. OUR APPROACH

The idea of our approach is based on the knowledge that an arrival curve can be conservatively approximated by a set of staircase functions [9], each of which can be modeled by a leaky-bucket kind event generator. Rather than generating events, we use the leaky bucket mechanism for online monitoring. In this context, the bucket capacity corresponds to the maximally tolerable number of bursty events and the leak rate models the period of the staircase function. The fill level of the bucket is used as an indicator for the remaining capacity of the tolerable burst.

For each staircase function, we employ two leaky buckets, namely *V*-bucket for input conformity verification and *R*-bucket for input regulation. The fill level of *V*-bucket indicates how many bursty incoming events can still be tolerated while the fill level of *R*-bucket shows how many bursty events are allowed to release. By simply tracking the fill levels of the buckets, the computationally expensive min-plus (de-)convolution normally used by the offline analysis can be eliminated for the online monitoring, resulting in a lightweight software or hardware implementation.

The flow of our approach is shown in Fig. 2. Upon each event arrival, the conformity of the event is tested. If the arrival of this event conforms to the specification, this event will be immediately released. If not, certain regulation is applied to enforce the conformity. For the current version of our algorithm, we delay the release time of non-conformed events. Discarding events due to deadline violation or buffer overflow can be easily adapted based on the proposed scheme. We will discuss their solutions in Section 4.3. Note that we only present the algorithm and proofs for the upper bound α^u. The conformity verification of the lower bound works in a similar manner.

4.1 Algorithm

Assume an arrival curve is approximated by n staircase functions S_i, $i \in n$. Each S_i is defined by a leaky bucket with two parameters, namely bucket capacity N_i^u and period δ_i^u. During runtime, the status of a bucket is tracked by two variables, namely the fill level *BFL* and a timer *CLK*. The timer *CLK* records the time passed within a period δ_i^u. Thus our algorithm maintains a 4-tuple $<BFL_i^v, CLK_i^v, BFL_i^r, CLK_i^r>$ during runtime, BFL_i^v, CLK_i^v for *V*-bucket and BFL_i^r, CLK_i^r for *R*-bucket. The algorithm is invoked when a signal comes. A signal can be triggered by the arrival of an event or the timeout of a timer. Initially,

Algorithm 1 On-the-fly traffic verification and regulation for an n-staircase arrival curve.

```
Input: signal $s$ ▷ tuple $<BFL_i^v, CLK_i^v, BFL_i^r, CLK_i^r>$ and
    event queue $q$ are global variables
1: for $i \leftarrow 1$ to $n$ do                              ▷ timeout
2:     if $s = CLK_i^v\_timeout$ then
3:         $BFL_i^v \leftarrow \min(BFL_i^v + 1, N_i^u)$
4:         reset_timer($CLK_i^v$)
5:     end if
6:     if $s = CLK_i^r\_timeout$ then
7:         $BFL_i^r \leftarrow \min(BFL_i^r + 1, N_i^u)$
8:         reset_timer($CLK_i^r$)
9:     end if
10: end for
11: if $s$ = event_arrival then                         ▷ event arrival
12:     for $i \leftarrow 1$ to $n$ do
13:         if $BFL_i^v = N_i^u$ then
14:             reset_timer($CLK_i^v$)
15:         end if
16:         $BFL_i^v \leftarrow BFL_i^v - 1$
17:     end for
18:     q.enqueue()
19: end if
20: if $\min_{i \in n}(BFL_i^v) < 0$ then                  ▷ nonconformity
21:     report_violation()
22: end if
23: while q.length()$>0 \wedge \min_{i \in n}(BFL_i^r) > 0$ do    ▷ regulation
24:     q.dequeue()
25:     for $i \leftarrow 1$ to $n$ do
26:         if $BFL_i^r = N_i^u$ then
27:             reset_timer($CLK_i^r$)
28:         end if
29:         $BFL_i^r \leftarrow BFL_i^r - 1$
30:     end for
31: end while
```

$BFL_i^v = BFL_i^r = N_i^u$ and $CLK_i^v = CLK_i^r = \delta_i^u$.

The pseudo code of the algorithm is shown in Algo. 1. BFL_i^v of *V*-bucket indicates the number of bursty events that can still be tolerated at current time. Its value is decreased by 1 when an event arrives (Line 16) and increased by 1 when CLK_i^v is timeout (Lines 2–5). BFL_i^v has a limit of N_i^u (Line 3). Based on the algorithm, BFL_i^v can be computed as a function of the trace R:

$$BFL_i^v(t) = \min(N_i^u + \lfloor \frac{t}{\delta_i^u} \rfloor - R(t),\, N_i^u) \qquad (5)$$

For any time interval $(s, t]$, BFL_i^v can be computed as:

$$BFL_i^v(t) = \min\left(BFL_i^v(s) + \lfloor \frac{t-s}{\delta_i^u} \rfloor - (R(t) - R(s)),\, N_i^u\right) \qquad (6)$$

Since BFL_i^v records the remaining capacity of the bucket, conformity violation occurs when $BFL_i^v < 0, \exists i \in n$ (Lines 20–22). In other words, a nonconformity occurs when the number of events arrived in the interval Δ is larger than $N_i^u + \lfloor \frac{\Delta}{\delta_i^u} \rfloor$.

The CLK_i^v of *V*-bucket notifies when the budget can be recharged. It is reset when BFL_i^v reaches its limit, i.e., $BFL_i^v = N_i^u$ (Lines 13–15). When BFL_i^v is equal to N_i^u, a burst of maximal N_i^u events can be tolerated from this time on. This case can also be considered as a renew point of the bucket. We will use this property in the later-on proofs.

The *R*-bucket works similarly. BFL_i^v controls when and how many events can be released (Lines 23–31). Only when all $BFL_i^r > 0$, an event can be released. It is decreased by 1 when an event is released (Line 29). Otherwise events will be postponed until every BFL_i^r turns nonzero. To release buffered events, we use a first-come first-out scheme.

4.2 Correctness

This section proves the correctness of our algorithm. For simplicity, we provide the formal proof for the case of $n = 1$, i.e., $\alpha^u(\Delta) = N^u + \lfloor \frac{\Delta}{\delta^u} \rfloor$. Proofs for $n > 1$ cases follow a similar scheme.

To prove the algorithm, we divide the time axis for the system execution into a set of consecutive time segments. A time segment is defined as follows.

Def. 1. *A time segment F is the time interval for the value of BFL^v changing from N^u back to N^u, i.e., between two renew points in the trace (Lines 2–5, Algo. 1).*

The starting and ending time instants for F_i are denoted by t_{S_i} and t_{E_i}, respectively. For the arrival of the n^{th} event e_n in the trace, we also designate t_n and x_n the time instant and the value of BFL^v, respectively. Based on above definitions, we have following lemmas.

Lemma 1. *Within any F_i, $i \in \mathbb{N}$, at time t_1, i.e., the first arrived event of this segment, $BFL^v = N^u - 1$. At each time instant t_i of the arrivals of subsequent events within F_i, $BFL^v \leq N^u - 2$.*

Proof. According to Def. 1, BFL^v always starts from N^u for any F_i. When the first event arrives, BFL^v turns to $N^u - 1$. The arrivals of subsequent events will further decrease BFL^v. Assume at some point of time, BFL^v is recharged back to $N^u - 1$ due to the timeouts of CLK^v (Lines 2–5). There are only two cases: a) a next timeout of CLK^v comes, BFL^v reaches N^u, and the segment ends, or b) an event arrives, BFL^v decreases back to $N^u - 2$. Therefore, the lemma holds. □

Lemma 2. *At time t_n for the arrival of event e_n, if $t_{m+n} - t_n \geq (m - (N^u - 1))\delta^u$, $\forall m$, the trace between $[t_n, t_{m+n}]$ conforms α^u.*

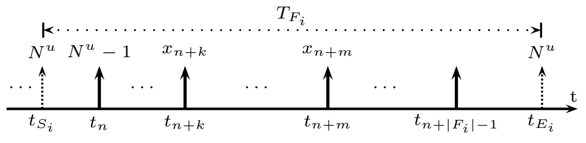
Proof. Consider an arbitrary interval $[s, t]$. There are $m + 1$ events arrived within this interval and these events are numbered as n, $n+1$, …, $m+n$, so that $t_{n-1} \leq s < t_n \leq t_{n+1} \leq \ldots \leq t_{m+n} \leq t$. We get $\alpha^u(t-s) \geq \alpha^u(t_{m+n} - t_n) = N^u + \lfloor \frac{t_{m+n} - t_n}{\delta^u} \rfloor \geq m + 1$. Because $R(t) = m + n$ and $R(s) = n-1$, we get $\alpha^u(t-s) \geq R(t) - R(s)$. From Eqn. (1), the lemma holds. □

Lemma 3. *Let T_{F_i} denote the length of segment F_i and $|F_i|$ the number of events arrived within F_i, $T_{F_i} \geq |F_i| \cdot \delta^u$.*

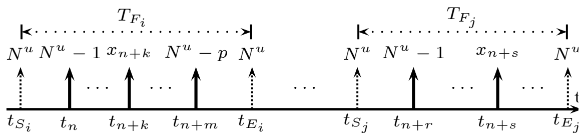
Proof. Fig. 3(a) illustrates such an example. According to the algorithm, the timer CLK^v is cleared at time instants t_n, t_{S_i}, and t_{E_i}. With Eqn. (6), we have $N^u - 1 + \frac{t_{E_i} - t_n}{\delta^u} - (|F_i| - 1) = N^u$. Therefore, $t_{E_i} - t_{S_i} \geq t_{E_i} - t_n = |F_i| \cdot \delta^u$. □

Theorem 1. *Given a trace R and a staircase α^u, in the case that $BFL^v \geq 0$, Algo. 1 guarantees R conform to α^u.*

Proof. What we need to prove is for $\forall\, 0 \leq s \leq t$, $R(t) - R(s) \leq \alpha^u(t - s)$. We consider two cases, i.e., s and t are located within one segment and in two different segments.



(a) Single-segment case.



(b) Multiple-segment case.

Figure 3: Graphical illustration of the proof.

First we consider the special case of $N^u = 1$. In this case, each segment allows only one event to guarantee $BFL^v \geq 0$, i.e., there is only one event in each segment. Without loss of generality, let s and t the arrival time of event e_n and e_m which arrive at segment F_n and F_m, respectively. Form Lem. 3, we have $t_{E_n} - t_n = \delta^u$ and $t_{S_m} - t_{E_n} = (m-1-n)\delta^u$. Thus $t_m - t_n = (t_m - t_{S_m}) + (t_{S_m} - t_{E_n}) + (t_{E_n} - t_n) \geq (m - n)\delta^u = (m - n - (N^u - 1))\delta^u$. Based on Lem. 2, the theorem holds for $N^u = 1$.

In the following, we provide the proof for $N^u \geq 2$.

• Single-segment case:

For any given segment F_i, assume e_n is the first event arrived in F_i, as shown in Fig. 3(a). Obviously, BFL^v is $N^u - 1$ at time instant t_n. We further assume that events e_{n+k} and e_{n+m} ($m > k \geq 0$) arrive within F_i at time instants t_{n+k} and t_{n+m}, the corresponding values of BFL^v being x_{n+k} and x_{n+m}, respectively. According to Eqn. (6), we have

$$N^u - 1 + \lfloor \frac{t_{m+n} - t_n}{\delta^u} \rfloor - m = x_{n+m} \quad (7)$$

$$N^u - 1 + \lfloor \frac{t_{n+k} - t_n}{\delta^u} \rfloor - k = x_{n+k} \quad (8)$$

For $k > 0$, we know $0 \leq x_{n+m}, x_{n+k} \leq N^u - 2$ (Lem. 1). Therefore, we have

$$\begin{aligned} t_{m+n} - t_{n+k} &= (t_{m+n} - t_n) - (t_{n+k} - t_n) \\ &= \lfloor \frac{t_{m+n} - t_n}{\delta^u} \rfloor \delta^u + \sigma_{m+n}\delta^u \\ &\quad - \lfloor \frac{t_{k+n} - t_n}{\delta^u} \rfloor \delta^u - \sigma_{k+n}\delta^u \\ &= (m + x_{n+m} - k - x_{n+k})\delta^u + (\sigma_{m+n} - \sigma_{k+n})\delta^u \\ &\geq (m - k - (N^u - 2))\delta^u + (\sigma_{m+n} - \sigma_{k+n})\delta^u \\ &\geq (m - k - (N^u - 1))\delta^u \end{aligned}$$

where[2] $\sigma_{m+n} = \frac{t_{m+n} - t_n}{\delta^u} - \lfloor \frac{t_{m+n} - t_n}{\delta^u} \rfloor$.

For $k = 0$, we have

$$\begin{aligned} t_{m+n} - t_n &= \lfloor \frac{t_{m+n} - t_n}{\delta^u} \rfloor \delta^u + \sigma_{m+n}\delta^u \\ &= (m + x_{n+m} - (N^u - 1))\delta^u + \sigma_{m+n}\delta^u \\ &\geq (m - (N^u - 1))\delta^u + \sigma_{m+n}\delta^u \\ &\geq (m - (N^u - 1))\delta^u \end{aligned}$$

Based on Lem. 2, the theorem holds for this case.

• Multiple-segment case:

We consider segments F_i and F_j with $j > i$. As shown in Fig. 3(b), event e_n is the first event in F_i with $BFL^v = N^u - 1$ and event e_{n+m} is the last event in F_j with $BFL^v = N^u - p$ ($p \geq 2$). In F_j, event e_{n+r} is the first event with

[2] σ_{k+n}, σ_{n+s}, and σ_{n+k} in the subsequent text follow similar definition.

$BFL^v = N^u - 1$. Therefore, there are $r - m - 1$ events arrived in the interval $[t_{E_i}, t_{S_j}]$. From Lem. 3, we have

$$t_{S_j} - t_{E_i} \geq (r - m - 1)\delta^u \quad (9)$$

$$t_{E_i} - t_n = (m + 1)\delta^u \quad (10)$$

Considering events e_{n+s} of F_j and e_{n+k} of F_i, we have following equations based on Eqn. (6):

$$N^u - 1 + \lfloor \frac{t_{n+s} - t_{n+r}}{\delta^u} \rfloor - (s - r) = x_{n+s} \quad (11)$$

$$N^u - 1 + \lfloor \frac{t_{n+k} - t_n}{\delta^u} \rfloor - k = x_{n+k} \quad (12)$$

For $k \neq 0, s \neq r$, we have $0 \leq x_{n+s}, x_{n+k} \leq N^u - 2$ (Lem. 1). Together with Eqns. (9)–(12), we get

$$\begin{aligned} t_{n+s} - t_{n+k} &= (t_{n+s} - t_{n+r}) + (t_{n+r} - t_{S_j}) + (t_{S_j} - t_{E_i}) \\ &\quad + (t_{E_i} - t_n) - (t_{n+k} - t_n) \\ &\geq (t_{n+s} - t_{n+r}) + (t_{S_j} - t_{E_i}) + (t_{E_i} - t_n) \\ &\quad - (t_{n+k} - t_n) \\ &\geq \lfloor \frac{t_{n+s} - t_{n+r}}{\delta^u} \rfloor \delta^u + \sigma_{n+s}\delta^u + (r - m - 1)\delta^u \\ &\quad + (m+1)\delta^u - (\lfloor \frac{t_{n+k} - t_n}{\delta^u} \rfloor \delta^u + \sigma_{n+k}\delta^u) \\ &= (x_{n+s} - x_{n+k} + s - k)\delta^u + (\sigma_{n+s} - \sigma_{n+k})\delta^u \\ &\geq (s - k - N^u + 1)\delta^u \end{aligned}$$

For $k \neq 0$, $s = r$, from Lem. 3, Eqns. (10), (9), and (12) as well as $N^u \geq 2$, we have

$$\begin{aligned} t_{n+r} - t_{n+k} &\geq (t_{S_j} - t_{E_i}) + (t_{E_i} - t_n) - (t_{n+k} - t_n) \\ &\geq (r - k + 1)\delta^u + (\sigma_{n+m} - \sigma_{n+k})\delta^u \\ &\geq (r - k - (N^u - 1))\delta^u \end{aligned}$$

For $k = 0$, we get $t_{n+s} - t_{n+r} \geq (s - r - N^u + 1)\delta^u$ and $t_{E_i} - t_n = (m+1)\delta^u$ from single-segment case and Eqn. (10). Then we have

$$\begin{aligned} t_{n+s} - t_n &\geq (t_{n+s} - t_{n+r}) + (t_{S_j} - t_{E_i}) + (t_{E_i} - t_n) \\ &\geq (s - N^u + 1)\delta^u \end{aligned}$$

From above cases, the theorem holds. □

Corollary 1. *At the time instant when BFL^v turns small than 0, a violation to α^u occurs.*

Proof. As shown in Fig. 3(a), assume $BFL^v < 0$ occurs when event e_{m+n} arrives in F_i, i.e., $x_{n+m} \leq -1$. According to Eqn. (7), $t_{m+n} - t_n = (m + x_{n+m} - (N^u - 1))\delta^u + \sigma_{m+n}\delta^u$. Consider the interval [s,t] with $s = t_n - \lambda$ and $t = t_{m+n}$, where $0 < \lambda < (1 - \sigma_{m+n})\delta^u$. Thus, $\alpha^u(t - s) = N^u + \lfloor \frac{t_{m+n} - t_n + \lambda}{\delta^u} \rfloor = m + x_{n+m} + 1 \leq m$. Because $R(t) = m + n$ and $R(s) = n - 1$, $\alpha^u(t - s) < R(t) - R(s)$. A violation to α^u occurs. □

Theorem 2. *The resulting trace regulated by Algo. 1 conforms to α^u.*

Proof. The output traffic is modulated by the *R*-bucket. The *R*-bucket works in the same mechanism as *V*-bucket. In addition, variable *backlog* (Lines 23) guarantees that BFL^r will never go below zero. Therefore, the theorem holds according to Thm. 1. □

4.3 Discussion

The algorithm and proof in the previous section are for the conformance verification and regulation of the upper bound of an arrival curve. Similar technique can be used for the violation detection of the lower bound. The regulation of input traffic to re-conform to a lower curve is however not possible in this context. As violation of the lower bound basically means no sufficient number of events occurs for a certain time interval, a regulation by injecting artificial events into the system would violate our basic assumption that we only consider time-invariant systems. Nevertheless, a warning can be issued to the application, so that the application itself might be able to react to the violation.

Note that another assumption of our approach is that the violated events will be stored in a buffer and released at a later point of time. Too many buffered events may lead to buffer overflow of our algorithm. Although it is unavailable, we nevertheless can detect such occurrence by modulating the event queue q in the algorithm. Another fact is that delaying the input events may result in deadline violation of input events. Detecting the deadline violations can also be included based on the current scheme.

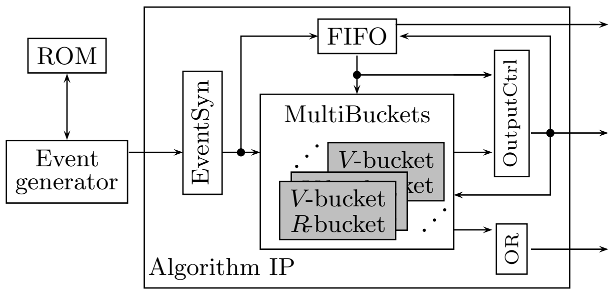


Figure 4: The block diagram of the FPGA testbed.

5. EXPERIMENTS

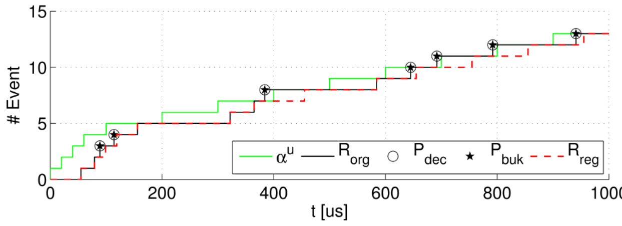
This section demonstrates the effectiveness of our approach by empirical case studies. We implement our algorithm both in Matlab and Verilog HDL. The Verilog HDL code is synthesized in ALTERA Cyclone III Fpga.

We adopt the PJD model (Section 3) for the specification of event streams. The upper bound α^u for such a model can be represented as the minimum of two staircase functions. The parameters of the two staircase functions can be computed as follows [9]:

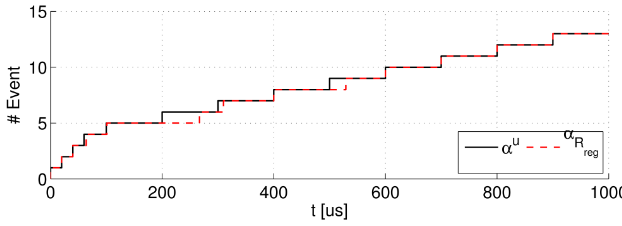
- Case $d = 0 \vee d \leq p - j$:
  $N^u = \lceil \frac{j}{p} \rceil + 1$; $N^l = -\lceil \frac{j}{p} \rceil$; $\delta^u = \delta^l = p$
- Case $d > 0 \wedge d > p - j$:
  $N_1^u = 1$; $\delta_1^u = d$; $N_2^u = \lceil \frac{j}{p} \rceil + 1$;
  $N^l = -\lceil \frac{j}{p} \rceil$; $\delta_2^u = \delta^l = p$

To generate traces with different patterns, the Rtc/Rts-toolbox [14] is used. We first generate a worst-case trace that conforms to the upper bound. Then we inject random events to artificially create violations. In our experiment, we employ an arrival curve with period of $100us$, jitter of $300us$, and delay of $20us$.

In order to validate our algorithm, we implement a discrete-time simulation in Matlab and an Fpga testbed. The Matlab simulation is implemented using the Rtc/Rts-toolbox. The testbed is comprised of an event generator IP and the algorithm IP, as the block diagram shown in Fig. 4. The event generator IP is used to generate events that comply with those used in the Matlab simulation. The algorithm IP itself consists of four modules. As shown in the figure, the MultiBuckets module contains a reconfigurable number of bucket pairs, each of which contains a *V*-bucket and *R*-bucket. The EventSyn module synchronizes the FIFO



(a) Detection and traffic regulation.



(b) Upper bound of the regulated trace.

Figure 5: Results for verification and regulation of a trace with seven violation points.

and MultiBuckets modules when events arrive. The output module controls the release of events. The FIFO module is used to buffer regulated events. The testbed is simulated using ModelSim. Details of the implementation are given in Fig. 6 in the appendix.

We compare the theoretical and experimental results in Fig. 5. The solid line R_{org} and dashed line R_{reg} represent the original and regulated traces, respectively. The star dots P_{dec} and round dots P_{buk} indicate the violation events computed by Eqn. (3) and detection by our algorithm, respectively. As expected, the two sets of dots match (Fig. 5(a)). We also compute the upper bound $\alpha_{R_{reg}}$ for the regulated trace R_{reg} (by Eqn. (3)) and compare with the input specification α^u. As the results shown in Fig. 5(b), $\alpha_{R_{reg}}$ is bounded by α^u. From Fig. 5, we experimentally confirm that our algorithm performs correctly.

We also report the resource consumption and latency for the Verilog HDL implementation. We synthesis the implementations for 2, 4, and 6 pairs of buckets using ALTERA Cyclone III EP3C120F780 development kit. The resource consumption is shown in Tab. 5. As shown in the table, the used logic elements even for the case of 6-pair buckets are still under 0.3% of the total resources (in total 119, 088 logic elements in the FPGA board). Furthermore, the resource usage is linear w.r.t the number of bucket pairs, which indicates our algorithm can be used to regulate the runtime traces for complex arrive curves. Note that, in this implementation, the same FIFO buffer is used for all buckets as events of the trace belong to the same arrival curve. Therefore, the size of the FIFO module is independent on the number of buckets and is decided by the over-burst events that is intended to tolerate.

Regarding timing overhead, 6 cycles are needed to transfer an event from the input to the output of our IP in the case that no regulation is employed. As the working frequency of the FPGA is set to 50 Mhz, this latency corresponds 120 ns. This result indicates the timing overhead of our algorithm is considerably small, which can be integrated into the WCET of the events without significant side-effects for the timing performance.

# Bucket Pair	Logic Elments	Register	LUTs	Memory (Bit)	Delay (cycle)
2	134	107	103	64	6
4	245	195	191	64	6
6	353	283	279	64	6

Table 1: Resource and timing overhead for the FPGA implementation.

6. CONCLUSION

This paper presents an online algorithm for the traffic conformity and regulation of hard real-time systems. Our algorithm can detect input violation and regulate the violated traffic to comply again with the specifications. We also present formal proofs, simulation results, and an FPGA implementation to demonstrate the effectiveness of our algorithm. The experiment results show that the resource and timing overheads of our algorithm are lightweight, particularly suitable for embedded systems with stringent resource constraints.

7. REFERENCES

[1] R. Alur and D. Dill. Automata for modeling real-time systems. In M. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer Berlin / Heidelberg, 1990. 10.1007/BFb0032042.

[2] C.-S. Chang. On deterministic traffic regulation and service guarantees: a systematic approach by filtering. *IEEE Transactions on Information Theory*, 44(3):1097–1110, may 1998.

[3] C.-S. Chang, R. Cruz, J.-Y. Le Boudec, and P. Thiran. A min, + system theory for constrained traffic regulation and dynamic service guarantees. *IEEE/ACM Transactions on Networking*, 10(6):805–817, dec 2002.

[4] R. L. Cruz and M. Taneja. An analysis of traffic clipping. In *Princeton University*, 1998.

[5] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano. MAST: Modeling and Analysis Suite for Real Time Applications. In *Proc. Euromicro Conference on Real-Time Systems*, pages 125–134, Delft, The Netherlands, June 2001.

[6] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System Level Performance Analysis — The SymTA/S Approach. *IEE Proceedings Computers and Digital Techniques*, 152(2):148–166, Mar. 2005.

[7] P. Kumar and L. Thiele. Cool shapers: Shaping real-time tasks for improved thermal guarantees. In *Proc. of Design Automation Conference (DAC)*, San Diego, 2011. ACM.

[8] K. Lampka, K. Huang, and J. J. Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *the International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[9] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems. *Design Automation for Embedded Systems*, 14(3):193–227, 2010.

[10] J. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.* Springer, 2001.

[11] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information Theory*, 44(3):1087–1096, may 1998.

[12] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.

[13] E. Wandeler, A. Maxiaguine, and L. Thiele. Performance analysis of greedy shapers in real-time systems. In *Design, Automation and Test in Europe (DATE)*, pages 444–449, Munich, Germany, 2006.

[14] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

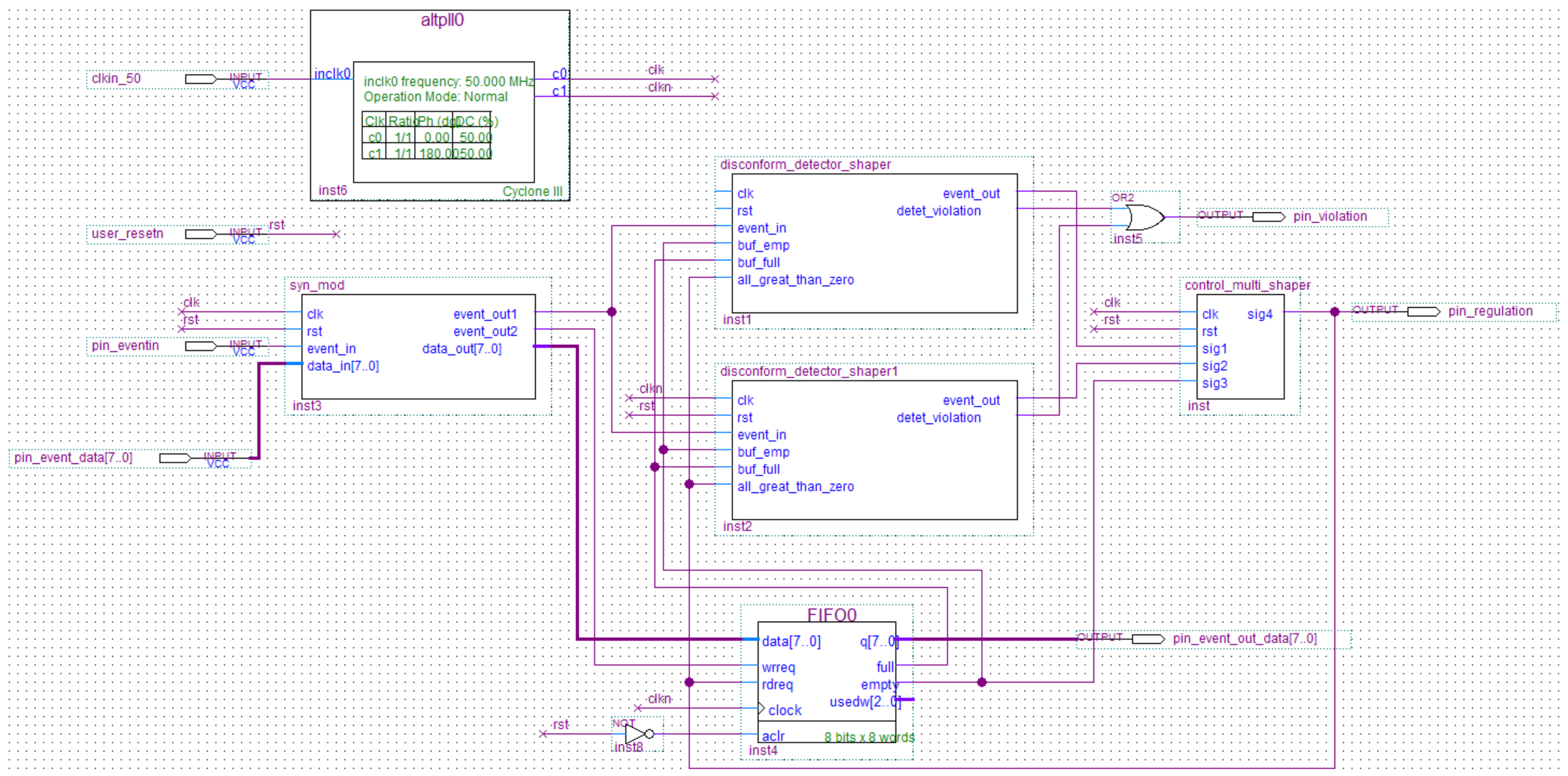


Figure 6: The detailed block diagram of algorithm IP (Fig. 4), implementing two bucket pairs. Pins `pin_eventin` and `pin_event_data`, respectively, are event synchronization signal and event data bus. Pin `pin_violation` represents a pulse signal when violation occurs. Regulation signals are generated at pin `pin_regulaion` and `pin_event_out_data`. When BFL_r value is larger than zero, `pin_regulation` asserts a high level and event data is put on `pin_event_out_data` at the same time.