Evaluation of Runtime Monitoring Methods for Real-Time Event Streams

Biao Hu[1],Kai Huang[1,2], Gang Chen[1], Alois Knoll[1]
[1]Tech. Univ. Muenchen TUM, [2]Sun Yat-sen University
[1]{hub,huangk,cheng,knoll}@in.tum.de, [2]huangk36@mail.sysu.edu.cn

Abstract— Runtime monitoring is of great importance as a safe guard to guarantee the correctness of system runtime behaviors. Two new methods, i.e., dynamic counters and l-repetitive function, are recently developed to tackle the runtime monitoring for hard real-time systems. This paper investigates in depth these two newly developed runtime monitoring methods, trying to evaluate and identify their strengths and weaknesses. Representative scenarios are used as our case studies to quantitatively demonstrate our comparisons. We also provide FPGA implementations and resource usages of both methods.

I Introduction

For the class of hard real-time embedded systems, meeting timing constraints, e.g., worst-case response time, is a fundamental requirement. Therefore, a large amount of research has been devoted to the design-time schedulability analysis at different abstraction levels. The resulting schedule of an analyzed system, on the other hand, relies on the assumption that all task activations and execution conform to the specifications used by the design-time analysis. But with the increasing complexity of embedded systems, runtime behaviors of tasks may not conform to the design-time specifications. For example, in heterogeneous distributed architectures, resources sharing or data-dependencies often result in unexpected execution sequences even when input tasks are activated strictly periodically [2]. Therefore, runtime monitoring is important to further guarantee the system timing properties obtained from the design-time analysis.

Two state-of-art online monitoring algorithms are recently developed, i.e., dynamic counters monitoring and l-repetitive function monitoring. Both monitoring algorithms are based on arrival curve model that can capture arbitrary event arrival patterns in the time interval domain. Dynamic counters monitoring assumes that an arrival curve can be conservatively approximated by a set of staircase functions, and each staircase function can be monitored by a counter [4]. Its validity has been mathematically proofed in [3]. The l-repetitive function assumes that an l-repetitive function can be constructed in a maximum busy-window period, then a history of most recent l events arrival time is kept to monitor coming events [7]. It is also a light-weight approach for monitoring arbitrary event streams. Although both monitoring algorithms are reported to be efficient, their monitoring differences are still unknown.

This paper investigates in depth these two newly developed runtime monitoring methods, trying to evaluate their performances and identify their strengths and weaknesses with respect to different system scenarios, and event arrival patterns. We consider it is important to know the differences in the modeling scope, the corresponding monitoring accuracy, and computation or memory overhead for these two methods, such that the techniques can be better used for specified systems. The detailed contributions are summarized as follows:

- We analyzed the differences of the dynamic counters [3, 4] and the l-repetitive function [7] for runtime monitoring of differently representative real-time system scenarios and event arrival patterns.
- It has to be noted here that authors in [7] claimed that it is not possible to monitor the pattern of periodic burst with dynamic counters. We developed a dynamic-counter based approach to monitor such pattern with low overheads.
- We prototyped hardware implementations on FPGA and presented FPGA resource usage for both methods.

The rest of this paper is structured as follows. We review related work in the next section. In Section III, basic background knowledge of the used techniques is described. In Section IV, we present the differences in modeling effects for the non-lossy and lossy real-time systems. Then we use standard and complex arrival curves to discuss their monitoring strengths and weaknesses. In Section V, FPGA experiments show resource overheads and timing latencies of both methods. Section VI concludes the paper.

II Related Work

Classically, in network communication systems, traffic shaper buffers the data packets of an incoming stream and delays them to conform to a specific arrival curve [5]. In real-time systems, shaper can also be used to regulate the system runtime behaviors. Wandeler *et al.* [13] analyzed the detailed buffer requirement and end-to-end delay of greedy shaper in two case systems. The analysis reveals the positive influence of greedy shaper on the system performance and buffer requirement. Richter *et al.* [10] proposed an event adaptation function to transform an event stream into another given stream. With the event adaptation function, buffering-size and buffering-delay calculation are automatically performed during adaptation. The drawback of all aforementioned work is the offline analysis. Compared to offline analysis, online monitoring is more useful in practical applications.

Except for online monitoring of dynamic counters or l-repetitive function, Moritz *et al.* [8] proposed a scheme for monitoring activation patterns of multiple streams in mixed-criticality real-time systems. This scheme provides a methodology, which is based on the interface design [14] and sensitivity analysis [9], to derive sets of activation pattern bounds for each monitored software component. However, the accuracy and efficiency of this methodology still rely on the single event stream monitoring method defined in Pareto interface tuples. Besides, authors in [6] also presented a monitoring scheme which monitors workload arrival functions for mixed-criticality systems. By specifying the cumulative worst-case workload of a group of streams through a workload arrival function, it is possible to increase the resource utilization while guaranteeing the system timing constraints. However, this monitoring scheme is used for monitoring low criticality tasks in a mixed-criticality system. For the high criticality tasks or strictly isolated tasks, it may be not effective. Though multi-mode monitoring or workload arrival function monitoring can increase the resource utilization, monitoring single task activation pattern is still significant for practical real-time system, because most of high safety systems isolate source event streams strictly [1]. In this paper, we only concentrate on monitoring single event stream of the real-time task.

III Background

In a network of multiple processors, tasks are interconnected among processors by a set of directed event streams. A task is activated by an incoming event. After the completion of a task, an event is produced at the task's outgoing event stream. During the design time, tasks are assumed to be executed within a designed extent. But during the runtime, the event streams should be monitored to guarantee that tasks are executed as designed.

A Dynamic counters monitoring

Event streams in a system can be described by a cumulative function $R(s,t)$, defined as the number of events seen in the time interval $[s,t)$. In Real Time Calculus [12], arrival curves with 2-tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ are used to describe the allowable maximum arrival events and minimum arrival events in the time interval of length Δ. It is expressed as follows:

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

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

Dynamic counters monitoring is based on the assumption that any monotone and time-invariant arrival curve can be conservatively approximated as the minimum on a set of staircase functions with the form $\alpha_i^u(\Delta) = N_i^u + \lfloor \frac{\Delta}{\delta_i^u} \rfloor$:

$$\forall \Delta \in \mathbb{R}_{\geq 0} : \alpha^u(\Delta) \leq \min_{i=1..n}(\alpha_i^u(\Delta)). \qquad (2)$$

A dynamic counter (DC_i) can be used to bound a single upper staircase function (α_i^u). The detail algorithm is shown in Algo. 1 [4]. Note that when an event arrives, all DC_i should be reduced one, and the exception is reported when any one of DC_i is less than 0.

The memory and computation overhead of Algo. 1 is decided by the number of used dynamic counters, and the error depends on how close the approximation is between $\alpha^u(\Delta)$ and $\min_{i=1..n}(\alpha_i^u(\Delta))$.

Algorithm 1 Implementing a dynamic counter to monitor a staircase function

```
Input:
    signal $s$, $\triangleright tuple < DC_i, CLK_i >$;
Output:
1:  if $s$ = event_arrival then
2:      if $DC_i = N_i^u$ then
3:          reset_timer($CLK_i$, $\delta_i^u$)
4:      end if
5:      $DC_i \leftarrow DC_i - 1$
6:  end if
7:  if $s = CLK_i\_timeout$ then
8:      $DC_i \leftarrow$ min($DC_i$+1,$N_i^u$)
9:      reset_timer($CLK_i$, $\delta_i^u$)
10: end if
11: if $DC_i < 0$ then
12:     report_exception
13: end if
```

B l-repetitive function monitoring

For discrete task activation patterns, arrival curves can also be described with a minimum distance function and maximum distance function. The minimum distance function $d^{min}(n)$ specifies the minimum distance among $n+1$ consecutive events in an event stream. And the maximum distance function $d^{max}(n)$ specifies the maximum distance among $n+1$ consecutive events in an event stream. For example, for periodic with jitter event models, we obtain

$$d^{min}(n) = max\{0, n * P - J\} \qquad (3)$$

$$d^{max}(n) = n * P + J, \qquad (4)$$

where P is the period and J is the jitter.

An l-repetitive distance function is a special minimum distance function that satisfies the following condition:

$$d(n) = \begin{cases} d_n(given), & n \leq l, \\ \max\limits_{\omega \in [1,l]} (d(\omega) + d(n-\omega)), & n > l. \end{cases} \qquad (5)$$

For such l-repetitive function, it has been shown that this l-repetitive function can conservatively approximate the upper arrival curve.

An assumption of l-repetitive function monitoring is that only a part of the d function is relevant to the verification of timing constraints. The relevant domain is $[1, n_{max}]$ where n_{max} is the number of task executions in the maximum system busy-window period. Then at most n_{max} task activations should be monitored in the system because pending task activations will only exist no longer than maximum system busy-window period [8]. The idea of l-repetitive function monitoring is to construct a more restrictive l-repetitive distance function with l events ($l \leq n_{max}$) to represent the distance function with n_{max} events. Then the monitoring overhead can be reduced to l. The monitoring algorithm can be seen in Algo. 2 [7].

Algorithm 2 Implementing l-repetitive function to monitor an event stream

```
Input:
    current time, trace buffer[$l$], $d[l]$;
Output:
1: for $i \in [0, l-1]$ do
2:     if current time - trace buffer[$i$] $< d[i]$ then
3:         report_exception
4:     end if
5: end for
6: right shift trace buffer
7: trace buffer[0] = current time
```

The computation and memory overhead of Algo. 2 is decided by the number of events in l-repetitive function, i.e., l. The error depends on how close the approximation is between $d^{min}(n)$ and the constructed l-repetitive function.

IV Monitoring algorithm comparison

As presented in the previous section, both monitoring methods need certain assumptions. The dynamic counters monitoring assumes that the arrival curve can be approximated by the minimum on a set of staircase functions, which indicates that arrival curve should be concave. On the other hand, the l-repetitive function monitoring assumes that the maximum number n_{max} of task activations can be obtained in the design-time analysis, and a conservatively l-repetitive function can be constructed to approximate d function. However, in some system scenarios, or for some arrival curves, the aforementioned assumptions are not valid.

In this section, we first analyze the monitoring differences of both methods in the non-lossy and lossy real-time system. Then, four standard and two special complex arrival curves are used to analyze the overhead and accuracy of the two monitoring methods.

A Real-time system scenarios

As depicted above, both monitoring methods can detect the violated event online under their respective assumptions. But how to deal with the violated events depends on what the system is. To the non-lossy real-time systems, the violated events cannot be discarded. Therefore, a buffer is needed to store the violated events, and to send the violated events out based on the designed arrival curve.

In [3], authors propose to use two dynamic counters to conform the runtime inputs as each designed staircase function. One conformity counter BFL_i^v is used to check the conformity of coming event, and another regulation counter BFL_i^r is used to regulate the event with a buffer. But actually, authors in [3] ignore a fact that the proposed conforming scheme only relies on regulation counter BFL_i^r. The conformity checking can also be done by BFL_i^r. A coming event is a violated event when $BFL_i^r = 0$, which means no event is allowed to arrive. We skip the proof because of page limit.

When the l-repetitive function monitoring is implemented in a non-lossy real-time system, Algo. 2 should use a buffer to store the violated events. The regulation scheme is that the earliest violated event in the buffer is sent out when current time satisfies the l-repetitive function. However, one assumption of Algo. 2 that the maximum busy-window period exists may be not true. The busy-window approach calculates the maximum time-window $\omega(q)$ during which the resource is busy processing until completion of the q-th instance of task τ_i. Thus, $\omega = \max\limits_{i,q}(\omega_i(q))$ is the longest time ω to execute tasks without becoming idle [7]. But in the non-lossy real-time systems, the violated events will interrupt the task execution. It is impossible to calculate the maximum busy-window period because violated events cannot be predicated. Thus, n_{max} may not exist. To get l without n_{max}, we propose to employ the concept of the sub-additive closure, which is known from Network Calculus [5].

Definition 1. *The sub-additive closure of a function α_l is given as*

$$closure(\alpha_l) = \inf_{n \geq 0}\{\alpha_l^{(n)}\} \tag{6}$$

where $\alpha_l^{(n)}$ denotes the function obtained by repeating (n-1) convolutions $\otimes$ of α_l with itself[1] *[5].*

Theorem 1.2.1 in [5] says that it is equivalent whether a trace is constrained through any wide-sense increasing arrival function or through the corresponding sub-additive closure, i.e., an event e_i satisfies $\alpha_l \Leftrightarrow e_i$ satisfies closure(α_l). This theorem indicates we can use a sub-additive closure of a limited segment to conservatively constrain the designed wide-sense arrival curve. To construct such a limited segment, l-repetitive arrival function that satisfies $closure(\alpha_l) \geq \alpha^u$ (α^u is the designed upper arrival curve) is redefined as follows

$$\alpha_l(\Delta) = \begin{cases} \alpha_l(\Delta)(given), & \Delta \leq d_l, \\ +\infty, & \Delta > d_l. \end{cases} \tag{7}$$

[1] $(f \otimes g)(\Delta) = \inf\limits_{0 \leq \lambda \leq \Delta}\{f(\Delta - \lambda) + g(\lambda)\}$

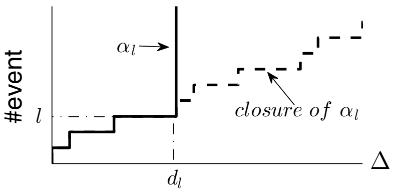


Fig. 1.: A limited l-repetitive arrival curve and its sub-additive closure

where $\alpha_l(\Delta)$ is sub-additive for $\Delta \leq d_l$, i.e., $\forall a, b, a + b \leq l: \alpha_l(a + b) \leq \alpha_l(a) + \alpha_l(b)$. As shown in Fig. 1, the closure of a limited l-repetitive arrival function α_l can represent a wide-sense increasing arrival curve.

In the lossy real-time systems, the violated events are discarded so that the non-violated events will not be interfered by the violated events. In this scenario, buffer is not necessary for a monitor. In Algo. 1, a little change that should be done is that DC_i is not reduced one by the violated event. In Algo. 2, n_{max} can be computed by the busy-window approach, and l-repetitive distance function can be constructed from n_{max} by the approximation approach introduced in [7].

In short, both algorithms can be applied in the non-lossy real-time systems with a support of buffer, and can be applied in lossy real-time systems without a buffer support. For l-repetitive function in non-lossy real-time systems, the assumption of busy-window period is invalid, then we propose to use the concept of sub-additive closure to construct l-repetitive function.

B Standard events pattern

The four standard event streams are strictly periodic events, sporadic events, periodic with jitter events, and periodic burst events [11]. Figure 2 shows their upper arrival curves. In this paper, we only concentrate on monitoring the upper arrival curve. Monitoring lower arrival curve is similar to monitor upper arrival curve.

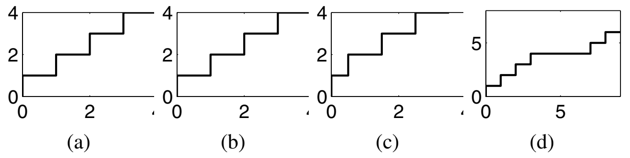


Fig. 2.: Upper arrival curves of four standard arrival curves. (a) Strictly periodic model; (b) Sporadic model; (c) Periodic with jitter model; (d) Periodic burst model.

From Fig. 2, we know the upper arrival curve of periodic events and sporadic events are the same pattern, and one staircase function or $l = 1$ repetitive function is enough to represent it. Periodic with jitter events model can be approximated as the periodic pattern, which however introduces some false negatives. It can be deduced that if the period of the approximated periodic events model is the same for the two methods in monitoring periodic with jitter model, the accuracy of both monitoring methods is also the same. The authors in [7] argue that dynamic counters algorithm is invalid to monitor periodic burst event pattern. However, for the standard periodic burst events, a scheme using dynamic counters can be effectively used to monitor it.

The standard periodic burst events model is characterized by three parameters, i.e., a minimum timing separation d between successive events, an interval δ that coming events number cannot exceed b, as is shown in Fig. 2(d). The upper arrival curve

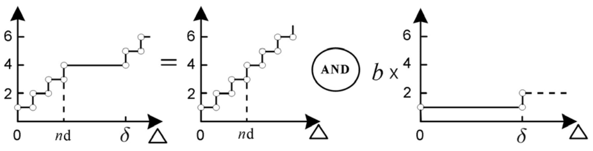


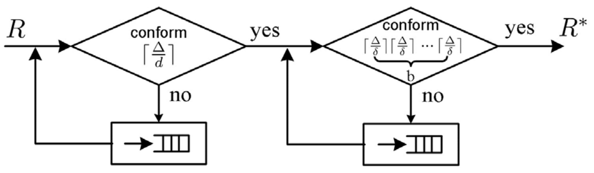
Fig. 3.: The diagram of periodic burst pattern equivalence.

can be expressed as follows [11],

$$\alpha^u(\Delta) = \lfloor \frac{\Delta}{\delta} \rfloor b + \min(\lceil \frac{\Delta - \lceil \frac{\Delta}{\delta} \rceil b}{d} \rceil, b) \qquad (8)$$

Periodic burst arrival curve cannot be approximated as the minimum of a set of staircase functions, but can be equivalent to a special logic composition of staircase functions.

As shown in Fig. 3, periodic burst is equivalent to a staircase function with a period of d, and the constraint of b staircase functions with period of δ. A counter with the period d and n counters with the period δ are used to guarantee that the minimal distance between two events, and the maximum events within any δ interval. The detail procedures are that, as shown in Fig. 4, we first check the arrival events conform α_d or not. If not, we buffer it. If yes, we check whether it conforms the b staircase functions. b staircase functions are assumed to be OR composition, i.e., any event is accepted if it does not violate any one of respective staircase functions [2]. If none counters in b staircase functions can accept an event, this event is violated. It will be delayed by buffer. Note that the two buffers in Fig. 4 are separated, and the buffer obeys first-in-first-out principle. The pseudo codes of this monitoring algorithm is shown in Algo. 3.



R
conform $\lceil \frac{\Delta}{d} \rceil$
yes
conform $\lceil \frac{\Delta}{\delta} \rceil \lceil \frac{\Delta}{\delta} \rceil \cdots \lceil \frac{\Delta}{\delta} \rceil$
b
yes
R^*
no
no

Fig. 4.: The flow of monitoring periodic burst arrival events.

We use a case to test the accuracy of this monitoring scheme. A periodic burst events stream specified as $b = 6$, $\delta = 180[\mu s]$, and $d = 20[\mu s]$ runs in a non-lossy real-time system. A matlab simulation result using this monitoring scheme is shown in Fig. 5. The solid curve α^u_{PB} is the designed arrival curve. The dash curve R is the actual events curve. The dash curve R^* is the regulated curve. The circles maker V are the violated events detected by the proposed monitoring scheme, and asterisks D are checked out by min-plus deconvolution[2]. For instance, events $(e_1,\ e_2,\ e_3)$ are the violated events, and they are delayed by buffer to be sent out at $(e_1^*,\ e_2^*,\ e_3^*)$. As expected, the V and D are the same, and the regulated curve R^* is lower than α^u_{PB}, which indicates the accuracy of this monitoring scheme.

To the l-repetitive function, keeping a history of most recent b events is sufficient to monitor this curve. Note that the difference of Algo. 3 and Algo. 2 in monitoring periodic burst events is that Algo. 2 does not require the minimal distance of successive events be the same, but Algo. 3 requires that. When monitoring a periodic burst events with the different minimal distance of successive events, an upper bound approximation with form of Eq. 8 can be used to represent this periodic burst arrival curve, and Algo. 3 can be applied to monitor it based on the approximated arrival curve. *In short, both monitoring methods can be used to monitor periodic or sporadic events without error, and can be used to monitor periodic with jitter events with the same error. For periodic burst events that dynamic counters cannot directly monitor, we develop a new scheme to monitor it.*

[2] $\sup_{u \geq 0}\{R(t+u) - R(u)\} = R(t) \oslash R(t) > \alpha^u(t)$

Algorithm 3 Online monitoring a periodic burst arrival events pattern.

Input:
signal s_d, s_δ, $tuple < DC^d, CLK^d >$, $tuple < DC_i^\delta, CLK_i^\delta >$ and event queue q_1, q_2;
Output:

```
1:  Algorithm 1 ← ($s_d$, $tuple < DC^d, CLK^d >$)
2:  Algorithm 1 ← ($s_\delta$, $tuple < DC_i^\delta, CLK_i^\delta >$)
3:  if report_exception($s_d$) then
4:      $q_1$.enqueue();
5:  end if
6:  if report_exception($s_\delta$) then
7:      $q_2$.enqueue();
8:  end if
9:  while $q_1$.length()>0 $\wedge$ $DC^d > 0$ do
10:     $q_1$.dequeue
11:     $DC^d = DC^d - 1$
12: end while
13: while $q_2$.length()>0 $\wedge$ $\max(DC_i^\delta) > 0$ do
14:     $q_2$.dequeue
15:     for $i \leftarrow 1$ to $n$ do
16:         if $DC_i^\delta > 0$ then
17:             $DC_i^\delta = DC_i^\delta - 1$
18:             break
19:         end if
20:     end for
21: end while
```

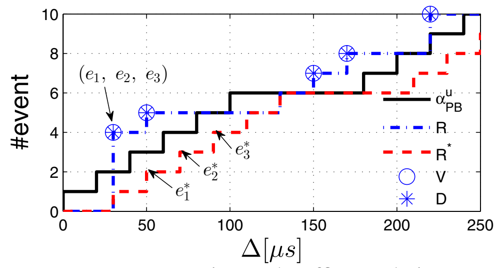


Fig. 5.: Detection and traffic regulation

C Complex arrival curves

From the above description, we may notice that dynamic counters monitoring is good at dealing with the type of arrival curve that can be conservatively approximated by staircase functions. l-repetitive function monitoring is good at dealing with the arrival curve that is segmentally repetitive. In this subsection, we discuss how to apply the two monitoring algorithms in monitoring an arrival curve marked Type 1 that is the minimum of a set of staircase functions, and another arrival curve marked Type 2 that is segmentally repetitive.

C.1 Complex arrival curve of Type 1

The arrival curve α_{min} of Type 1 is shown in Fig. 6, where $\alpha_{min} = \min(\alpha_1, \alpha_2, \alpha_3)$, $\alpha_1 = N_1^u + \lfloor \frac{\Delta}{\delta_1} \rfloor$, $\alpha_2 = N_2^u + \lfloor \frac{\Delta}{\delta_2} \rfloor$, and $\alpha_3 = N_3^u + \lfloor \frac{\Delta}{\delta_3} \rfloor$ ($N_1^u < N_2^u < N_3^u$, $\delta_1 < \delta_2 < \delta_3$). We can use three dynamic counters to monitor it in full accuracy, whenever in non-lossy or lossy real-time system.

For the l-repetitive function monitoring, l-repetitive minimal

TABLE I

: The error and overhead of both monitoring methods in different scenarios

Monitor	Dynamic Counters				l-repetitive function			
Arrival Curve	Type 1		Type 2		Type 1		Type 2	
Property	error	overheads	error	overheads	error	overheads	error	overheads
Non-lossy system	0	# DC	$\alpha_l - \alpha_{min}$	# DC	$\alpha_{min} - \alpha_l$	$\{l \mid l = \frac{d_l}{\delta_{max}}\}$	0	# events
Lossy system	0	# DC	$\alpha_l - \alpha_{min}$	# DC	$\alpha_{min} - \alpha_l$	$\{l \mid l = \frac{d_l}{\delta_{max}}\}$	0	# events

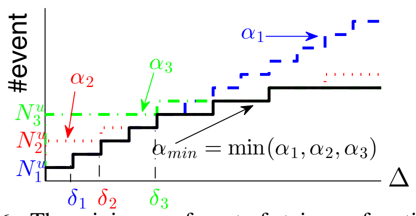


Fig. 6.: The minimum of a set of staircase functions

distance is different in different real-time system scenarios. For the non-lossy real-time system, a repetitive segment α_l with l and d_l (Eq. 7 and Fig. 1) is assumed to be constructed. In a case that $\Delta = k \cdot d_l (k \in N, k = +\infty)$, we have $\min\limits_{i=1..n}\{\alpha_i\}(\Delta) = N_x^u + \lfloor \frac{\Delta}{\delta_{max}} \rfloor$, where $\delta_{max} = \max\limits_{i=1..n}\{\delta_i\}$ (in this case, $\delta_{max} = \delta_3$, $N_x^u = N_3^u$), and

$$closure(\alpha_l(\Delta)) - \min_{i=1..n}\{\alpha_i\}(\Delta) \\ = \frac{\Delta}{d_l} l - N_x^u - \lfloor \frac{\Delta}{\delta_{max}} \rfloor = k \cdot l - N_x^u - \lfloor \frac{k \cdot d_l}{\delta_{max}} \rfloor \quad . \tag{9}$$

We can get,

$$\lim_{k=+\infty} (closure(\alpha_l(\Delta)) - \min_{i=1..n}\{\alpha_i\}(\Delta)) = \\ \begin{cases} +\infty, & d_l < l \cdot \delta_{max} \\ constant, & d_l = l \cdot \delta_{max} \\ -\infty, & d_l > l \cdot \delta_{max} \end{cases} \tag{10}$$

The offset between $closure(\alpha_l)$ and α_{min} should not increase with the increasing of k. Otherwise, the monitor performance will become worse and worse with increasing number of events. So a requirement of d_l and l is $d_l = l \cdot \delta_{max}$. Then l-repetitive function will keep a limited error in monitoring arrival curve of Type 1.

In lossy real-time systems, there exists n_{max} that keeping most recent n_{max} events arrival time is enough to monitor coming events, and there is no error. If a more restrictive segment with l events (less than n_{max} events) is constructed, the computation and memory overhead is reduced, but an extra error is also introduced because Type 1 arrival curve cannot be fully accurately approximated by an l-repetitive function ($l \leq n_{max}$).

In short, to monitor Type 1 arrival curve, the overhead of l-repetitive function algorithm is l that satisfies $d_l = l \cdot \delta_{max}$ in non-lossy real-time systems, and there is a limited error. In the lossy real-time systems, the overhead is n_{max}, and there is no error. The overhead of dynamic counters monitoring is the number of used dynamic counters, and there is no error, whenever in the non-lossy or lossy real-time systems.

C.2 Complex arrival curve of Type 2

The second arrival curve α_l is segmentally repetitive, and each segment is sub-additive, as shown in Fig. 7. The length of the repetitive segment is d_l, which contains l events. With the theorem 1.2.1 in [5], the closure of a segment with the length of l is equivalent to the whole arrival curve. Therefore, monitoring most recent l events is enough to monitor the whole arrival curve. From this figure, we can find that the interval between two different events is different. We assume the minimum of a set of staircase functions ($\min\limits_{i=1..n}\{\alpha_i\}$, $\alpha_i = N_i^u + \lfloor \frac{\Delta}{\delta_i} \rfloor$) are used to approximate α_l, where $\delta_{max} = \max\limits_{i=1..n}\{\delta_i\}$. In the case that $\Delta = k \cdot d_l$ ($k \in N, k = +\infty$), we have $\min\limits_{i=1..n}\{\alpha_i\} = N_x^u + \lfloor \frac{\Delta}{\delta_{max}} \rfloor$, and we can get the same conclusion with Type 1 arrival curve analysis, i.e., $d_l = l \cdot \delta_{max}$ is required to guarantee that monitoring error is limited with increasing number of monitored events.

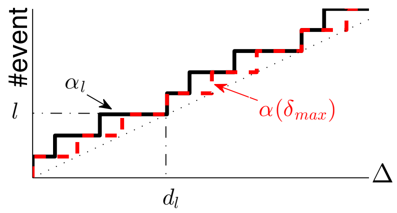


Fig. 7.: The segmentally repetitive curve

Therefore, to guarantee monitoring performance will not decrease with increasing number of events in the non-lossy real-time systems, it is required that the maximum period in this set of staircase functions satisfies $d_l = l \cdot \delta_{max}$, as shown in Fig. 7. In the lossy real-time systems, the set of staircase functions can be obtained based on maximum busy-window period. However, for l-repetitive function monitoring, keeping the arrival time of most recent l events is enough to monitor this segmentally repetitive curve without error, whenever in the non-lossy or lossy real-time systems. The monitoring differences of two types arrival curves are summarized in Tab. I.

V Implementation Overheads Evaluation

In this section, we implement both online monitoring methods with FPGA in the non-lossy real-time systems, and evaluate their effectiveness with synthesis report.

As shown in Fig. 8, the monitoring FPGA IP contains four modules [3]. The EventSyn module checks the coming event signal and data, and transmit them to FIFO. The FIFO module is used to buffer regulated events. The OutPutCtrl module control the release of data. Dynamic Counter module or l-repetitive module is used to regulate the output of event streams. In ModelSim, the coming events and data are generated with an event generator. For both approaches, we set the timer 16 bits. For l-repetitive monitoring, we have to initialize the minimal distance. Since the frequency of the processor is 50 Mhz, the tick is $0.02\mu s$. Assume the length of repetitive segment d_l defined in Eq. 7 is $500\mu s$, the initial minimal distance should be 16 bits to keep this events arrival trace.

After testing the correctness of our Verilog HDL code in ModelSim, we synthesize the implementations in Quartus using ALTERA Cyclone III EP3C120F780 device. In this test, we only concentrate on the resource overhead of each monitoring method, and ignore their accuracies. For l-repetitive monitoring scheme, we choose six l values. l values are 5 to 30 with increment of 5. For dynamic counters monitoring, we choose six dynamic counters to be 1 to 6 with increment of 1. As shown in Fig. 9, we list the logic elements, registers and logic array blocks (LABs) as three resource usage metrics.

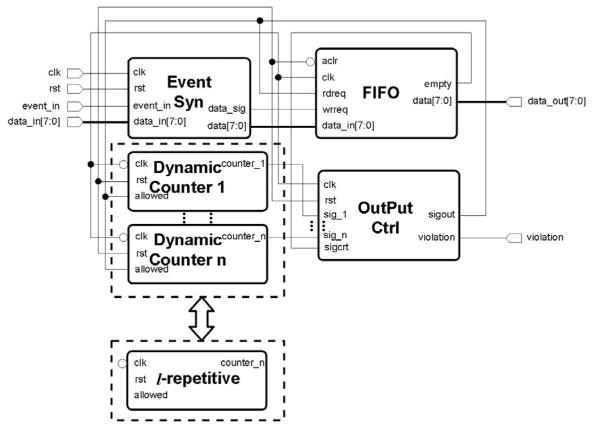


Fig. 8.: Block diagram of the FPGA testbed

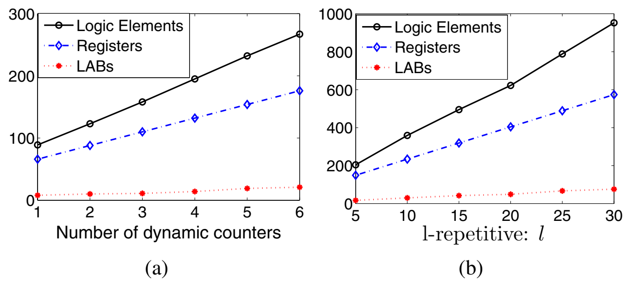


Fig. 9.: Results of resource usage after synthesis. (a) Dynamic counters monitoring; (b) l-repetitive monitoring.

From this figure, we can see that resource usage is linear with increasing l or number of dynamic counters. Resource usage of 5-repetitive monitoring is almost the same with 4 dynamic counters, which counts less than 0.3% of the total resources (total logic elements is 119,008 in this FPGA board).

We are also concerned about the influence of the d_l on the resource usage. In a non-lossy realtime system, we use 4 dynamic counters or 5-repetitive function to monitor a Type 2 arrival curve. In $d_l = l \cdot \delta_{max}$, l is fixed to be 5, and δ_{max} will be positively monotone with d_l. We set d_l to be [10^0, 10^1, 10^2, 10^3, 10^4, 10^5] μs, then the repetitive minimum distance and δ_{max} in dynamic counters monitoring should be 6, 9, 13, 16, 19, 23 bits. The resource usages are shown in Fig. 10. From these two figures, we can find that the resource overhead of dynamic counters monitoring changes not much with the increasing of d_l. However, the resource overhead of l-repetitive function monitoring changes more obviously than dynamic counters monitoring, but still keeps a low level.

From the above resource usage comparisons of two monitoring methods, we can find that resource overhead is more sensitive to dynamic counters number or the events number in l-repetitive function, and not sensitive to the length of one repetitive segment.

As for the monitoring timing latency, 5 clock cycles are needed to transfer an event from the input to the output in the case that no data is regulated. The timing latency is the same in both monitoring methods, and keeps constant with different l or number of dynamic counters because programs run in parallel in FPGA hardware. This result indicates that timing overhead of both monitoring methods is considerably small, which have little influence on system timing behavior.

VI Conclusions

This paper presents evaluations of the two latest developed runtime monitoring techniques. We demonstrate the strengths and weaknesses of these two techniques in different real-time

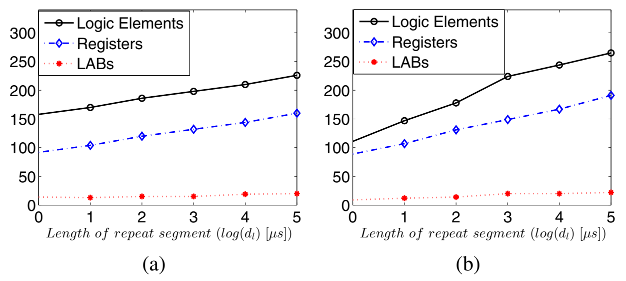


Fig. 10.: Resource usage with increasing busy-window period. (a) Dynamic counters monitoring; (b) l-repetitive monitoring.

scenarios, and provide the accuracy and overhead analysis with different arrival curves. By implementing both methods as hardware IP on FPGAs, we also provide comparisons on the level of hardware resources.

Acknowledgments

This work has been partly funded by German BMBF projects ECU (grant number: 13N11936) and Car2X (grant number: 13N11933).

References

[1] Do-178b - software considerations in airborne systems and equipment certification. Dec 1992.

[2] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the symta/s approach. *Computers and Digital Techniques*, 152(2):148–166, Mar 2005.

[3] K. Huang, G. Chen, C. Buckl, and A. Knoll. Conforming the runtime inputs for hard real-time embedded systems. In *Design Automation Conference*, pages 430–436, 2012.

[4] K. Lampka, K. Huang, and J.-J. Chen. Dynamic counters and the efficient and effective online power management of embedded real-time systems. In *International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS, pages 267–276. ACM, 2011.

[5] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer, 2001.

[6] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst. Monitoring of workload arrival functions for mixed-criticality systems. In *Real-Time Systems Symposium*, pages 88–96, Dec 2013.

[7] M. Neukirchner, T. Michaels, P. Axer, S. Quinton, and R. Ernst. Monitoring arbitrary activation patterns in real-time systems. In *Real-Time Systems Symposium*, pages 293–302, Dec 2012.

[8] M. Neukirchner, S. Quinton, R. Ernst, and K. Lampka. Multi-mode monitoring for mixed-criticality real-time systems. In *Hardware/Software Codesign and System Synthesis*, CODES+ISSS, pages 1–10, Sept 2013.

[9] M. Neukirchner, S. Quinton, T. Michaels, P. Axer, and R. Ernst. Sensitivity analysis for arbitrary activation patterns in real-time systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 135–140, March 2013.

[10] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *Computer*, 36(4):60–67, April 2003.

[11] K. Richter, R. Racu, and R. Ernst. Scheduling analysis integration for heterogeneous multiprocessor soc. In *Real-Time Systems Symposium*, pages 236–245, Dec 2003.

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

[13] E. Wandeler, A. Maxiaguine, and L. Thiele. On the use of greedy shapers in real-time embedded systems. *ACM Trans. Embed. Comput. Syst.*, 11(1):1:1–1:22, Apr. 2012.

[14] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Proceedings of the 5th ACM international conference on Embedded software*, pages 80–89. ACM, 2005.