# Performance Analysis of Reconfiguration in Adaptive Real-Time Streaming Applications

Jun Zhu, Ingo Sander and Axel Jantsch
Royal Institute of Technology, Stockholm, Sweden
{junz, ingo, axel}@kth.se

## Abstract

*We propose a design optimization framework for adaptive real-time streaming applications. The main contribution is a hybrid approach for performance analysis combining formal analysis and simulation using a two-phase framework. We formulate the scheduling problem of adaptive streaming applications with ILP analysis, and use the simulation based on the synchronous model of computation to ensure throughput guarantees. We finally illustrate the capabilities of our methodology by experiments.*

## 1. Introduction

Partially reconfigurable FPGAs allow to dynamically reprogram a part of the FPGA on the fly, while the rest continues its operation without being affected. This feature enhances the adaptive capability of many embedded signal processing and multi-media streaming applications working in dynamic environments.

However, this combination of flexibility and efficiency does not come for free. Adaptivity adds another dimension of complexity to the design process, while the system performance during reconfiguration still needs to be satisfied. Thus, to ensure the system's performance during reconfiguration, without losing efficiency, is a key challenge for future design methodologies to exploit the full potential of adaptivity.

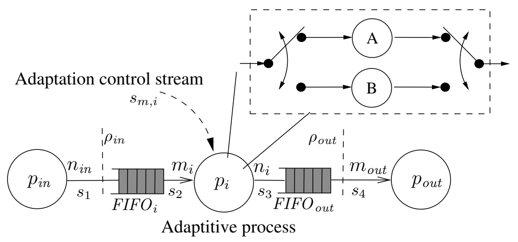


Figure 1: A minimal adaptive streaming application model

A minimal adaptive streaming application model is illustrated in Figure 1, which we use as a tutorial example in this paper. Nodes denote the computation processes. Edges associated with FIFOs denote the communication channels with finite storage, which decouples the input data streams from output data streams of each communication channel (e.g. $FIFO_i$ decouples $s_1$ from $s_2$).

Processes read tokens from the input-side FIFOs, and emit the result data tokens to the output-side FIFOs at the end of the computation. The input/output token numbers are fixed [7] and denoted as symbols at each side of the communication channels, e.g. process $p_i$ has $m_i$ input tokens and $n_i$ output tokens. Meanwhile, the adaptive process $p_i$ responds to the adaptation control stream $s_{m,i}$, and can switch between two different working modes A and B, as shown in the dashed box. While the stream source $p_{in}$ provides a peak data rate $\rho_{in}$ (on the communication channel cutting by the dashed-line), an average output data rate $\rho_{out}$ needs to be guaranteed by the application even during the reconfiguration of process $p_i$. The adaptation control signal $s_{m,i}$ might either come from an external controller or be retrieved from the data streams.

We propose a hybrid approach for performance analysis combining formal analysis and simulation using a two-phase framework. We implement our framework and use several adaptation scenarios on both the tutorial and industrial applications to show its capabilities.

The rest of the paper is structured as follows. Section 2 discusses the related work. We introduce the architecture model of our target platform, a reconfigurable FPGA in Section 3. Section 4 introduces the formal models we use for steaming applications. Section 5 proposes our framework for adaptive systems and its implementation. Section 6 shows the experimental results. Finally, Section 7 concludes the paper.

## 2. Related Work

Models of computations have been widely used as a formal base in streaming applications design. Synchronous data flow (SDF) [7] is used to describe the functionalities, and schedule applications with bounded buffer sizes. To provide timing guarantees, Govindarajan et al. [5] and Stuijk et al. [10] apply a timed SDF model for buffer requirement minimization with performance guarantees.

The synchronous model of computation has been very successful in the context of industrial safety-critical real-time systems [1]. We adopt this model in the recent work [8] for the modelling of adaptive systems. However, design efficiency of adaptive real-time embedded systems has not been addressed.

Network calculus [2] and real-time calculus (RTC) [3] are both a collection of methods in deterministic queuing theories. They formalize the incoming workloads and processing capabilities as cumulative functions of time, and suit system analysis of performance guarantees and buffer dimensioning. Although RTC further extends network calculus from network domain to the real-time embedded system domain, none of them takes adaptive systems into consideration.

Our approach in this paper is close to the spirit of a hybrid method in [6], which combines RTC formal analysis with cycle accurate simulation to speed up performance analysis. Our approach is different in that it uses formal semantics in both integer linear programming (ILP) and the synchronous model based simulation, and targets the design efficiency of adaptive real-time embedded systems, which has not yet been covered by others work to the extend of our knowledge.

## 3. Architecture Model

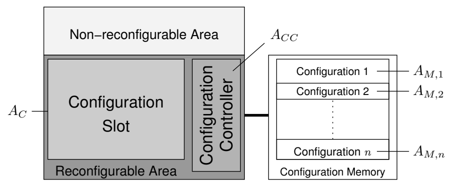


Figure 2: Overview of a reconfigurable FPGA using just-in-time reconfiguration with a single configuration slot.

Our target architecture is a partially reconfigurable FPGA as illustrated in Figure 2. For discussion, we divide the configuration-related FPGA area (excluding the non-reconfigurable area) into three parts.

The *reconfigurable area* is the space for configurations that are only needed for a limited amount of time at run-time. It can be used to store several configurations at the same time. However, here we focus on a "just-in-time"-approach (JIT), which uses a single configuration slot. Every time a system function is needed, the function is loaded into the reconfiguration slot. The *configuration memory* is used to store all configurations of different working modes in a compressed format (with the ratio $k_C$) that can be loaded into the reconfigurable area at run-time. It can either be a local memory, or an external memory (Flash, DDRAM, SRAM, etc.). The *configuration controller* enables reconfiguration. The only unit we use for area is logic elements (LE). Area requirements in form of memory elements are converted into logic elements.

During reconfiguration a bit-stream is loaded from the configuration memory into a part of the reconfigurable area. The *reconfiguration time* $t_{R,i}$ is both technology and stream-size dependent. To abstract from technology details, we assume that $t_{R,i}$ grows linearly with the size of the bit stream $A_{M,i}$ of the configuration $i$, thus $t_{R,i} = k_R A_{M,i}$. The configuration time $t_{R,i}$ lies typically in the area of milliseconds and can not be neglected. The consequence may be that there is a need for extra buffers with area $A_{Buffer}$, which includes the output buffer to sustain the output rate during reconfiguration and possibly an extra buffer to store input data tokens.

The total area cost for the JIT configuration is

$$A_{JIT}=A_{CC}+\sum_{i=1}^{i=n} A_{M,i}+k_C\cdot max(A_{M,1},...,A_{M,n})+A_{Buffer} \quad (1)$$

## 4 Formal models on streaming applications

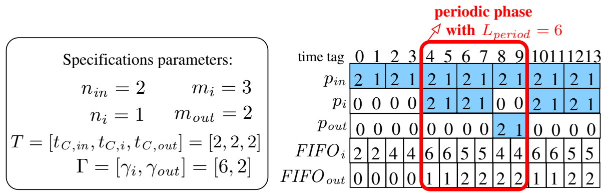
In this section, we illustrate two formal models on streaming application. One is the synchronous model (in Section 4.1), which provides a simulation mechanism and performance guarantees for regular (non-adaptive) streaming applications. The other is the ILP formulation (in Section 4.2), which fits well the changing working modes of adaptive systems but is in lack of performance guarantees. In this paper, we assume the time period always starts with 0.

### 4.1 Synchronous model

In the synchronous model, time slots are numbered with $n \in \mathbb{N}_0$. The data stream $s$ is a time indexed set of events, $s = \{e_0, e_1, \cdots, e_n, \cdots\}$. Each event $e_n = (n, v_n)$ represents the number $v_n \in \mathbb{N}_0$ of data tokens present during the time slot $n$.

For an application model, the process computation *latency* list $T$ contains computation time $t_{C,i}$ to execute each process $p_i$ once. The storage capabilities are captured in a FIFO *size* list $\Gamma$, which contains the storage capacity $\gamma_i$ in data tokens for each $FIFO_i$. For instance, the example application in Figure 1 has $T = [t_{C,in}, t_{C,i}, t_{C,out}]$ and $\Gamma = [\gamma_i, \gamma_{out}]$.

We use self-timed scheduling [9], which means a process *executes* only when the input-side FIFOs have sufficient data tokens and the output-side FIFOs have enough vacant space. In the scheduling test, a process $p_i$ needs to demand and reserve a vacant space $d_{i,t} \in \{0, n_i\}$ at time tag $t$ from the output-side FIFOs. While a process is computing, the data tokens remain on the input-side FIFOs until the computation is completed.



Specifications parameters:
$n_{in} = 2$ $m_i = 3$
$n_i = 1$ $m_{out} = 2$
$T = [t_{C,in}, t_{C,i}, t_{C,out}] = [2, 2, 2]$
$\Gamma = [\gamma_i, \gamma_{out}] = [6, 2]$
periodic phase with $L_{period} = 6$
time tag 0 1 2 3 4 5 6 7 8 9 10 11 12 13
$p_{in}$ 2 1 2 1 2 1 2 1 2 1 2 1 2 1
$p_i$ 0 0 0 0 2 1 2 1 0 0 2 1 2 1
$p_{out}$ 0 0 0 0 0 0 0 0 2 1 0 0 0 0
$FIFO_i$ 2 2 4 4 6 6 5 5 4 4 6 6 5 5
$FIFO_{out}$ 0 0 0 0 1 1 2 2 2 2 1 1 2 2

Figure 3: A self-timed schedule of the example application with specified specification parameters

The example application is instantiated as a non-adaptive system (i.e. $s_{m,i}$ is neglected) with the parameters on the left side of Figure 3, which implies an input peak data rate $\rho_{in} = \frac{n_{in}}{t_{in}} = 1$ to process $p_i$. The corresponding self-timed schedule is shown on the right side, in which the process and FIFO status are listed in separated rows. The time evolution is depicted in corresponding columns and advances 1 per column. At each time tag, a process in *executing* (shadowed) state has a number to denote the remaining execution time slots, a *stalling* (non-shadowed) process status is denoted as 0, and a FIFO status is denoted as the occupied storage space in tokens. As the schedule advances to time tag 10, the application encounters the same process and FIFO status list ($T'$ and $\Gamma'$) as at time tag 4 (i.e. $T'_{10} = T'_4 = [2, 2, 0]$ and $\Gamma'_{10} = \Gamma'_4 = [6, 1]$), and enters a periodic phase. The periodic phase has length $L_{period} = 6$, in which the sink process $p_{out}$ always runs 1 time. Thus, process $p_i$ guarantees an average output data rate $\rho_{out} = \frac{1 \cdot m_{out}}{L_{period}} = \frac{1}{3}$.

## 4.2 ILP analysis of streaming applications

We construct our event model as cumulative functions on the synchronous data streams (Section 4.1), which is similar to [4, 3]. We call the combination of a process $p_i$ and its adjacent input-side FIFO(s) $FIFO_i$ a processing resource $pr_i$.

We assume without loss of generality that a specified processing resource $pr_i$ has the adjacent preceding processing resource $pr_{in}$ and adjacent succeeding processing resource $pr_{out}$, as illustrated in Figure 1. The input/output workloads and processing capabilities of $pr_i$ are characterized as the following cumulative functions:

**Definition 1** *(Arrival function) The arrival function $R_i(t)$ of the processing resource $pr_i$ is defined as the sum of tokens arriving from the input data stream(s) (e.g. $s_1$ in Figure 1) during the time interval $[0, t], t \in \mathbb{N}_0$.*

**Definition 2** *(Output function) The output function $R'_i(t)$ of the processing resource $pr_i$ equals to the arrival function of the adjacent succeeding processing resource $pr_{out}$, thus $R'_i(t) \equiv R_{out}(t)$.*

**Definition 3** *(Service function) The service function $C_i[s, t]$ of the processing resource $pr_i$ is defined as the sum of tokens served by $p_i$ and removed from the buffer $FIFO_i$ via the data stream(s) (e.g. $s_2$ in Figure 1) during the time interval $[0, t], t \in \mathbb{N}_0$.*

To describe the extra buffer demanding and reservation in the scheduling test (see Section 4.1), we define

**Definition 4** *(Demand function) For the processing resource $pr_i$ with the output function $R'_i(t)$, the demand function $D_i(t)$ is defined as the sum of $R'_i(t)$ and the demanding space $d_{i,t}$ at time tag $t$ on the FIFO(s) (e.g. $FIFO_{out}$ in Figure 1) of the adjacent succeeding processing resource $pr_{out}$, i.e. $D_i(t) = R'_i(t) + d_{i,t}, d_{i,t} \in \{0, n_i\}$.*

We derive the backlog $B_i(t)$ (tokens received but not served) in $FIFO_i$ of the processing resource $pr_i$ to be

$$B_i(t) = R_i(t) - C_i(t), \quad \forall t \in \mathbb{N}_0 \tag{2}$$

In self-timed scheduling, the buffer space required $B'_{out}(t)$ on the output-side FIFO $FIFO_{out}$ of the processing resource $pr_i$ (equals to $B_{out}(t) + d_{i,t}$) is

$$B'_{out}(t) = D_i(t) - C_{out}(t), \quad \forall t \in \mathbb{N}_0 \tag{3}$$

We assume the JIT configuration of the adaptive process $p_i$ takes $t_{R,i}$ time and two succeeding configurations have the minimal interval $t_{interR,i}$. We formalize the semantics of the streaming application in self-timed scheduling with the ILP formulation, with the full list of constraints given in the Appendix. The ILP analysis can formulate the design requirements as objective functions subject to different constraints (changing scenarios), which fits well the varying working modes of adaptive systems. However, the ILP analysis can only provide guarantees within the upper-bound time $t$ considered for the Constraints (in Appendix), thus might lead to optimistic buffer dimensioning.

## 5 Framework and implementation

We aim at a hybrid approach for adaptive systems with performance guarantees. A generic-framework, which combines ILP analysis and the simulation in the synchronous model of computation, is illustrated in Figure 4. The inputs to the framework are the streaming application specifications (e.g. application model, adaptation strategy, specification parameters, $\rho_{out}$, etc). The workflow can be described by the following steps:

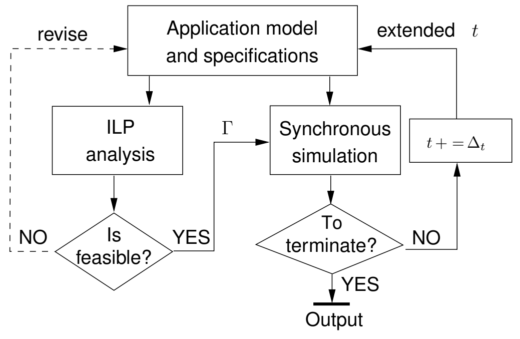


Figure 4: Generic framework

**Step 1:** When the problem is feasible, the ILP analysis outputs the buffer $\Gamma$ required for the given specifications; otherwise, the specifications need to be revised (which is out of the scope of this paper).

**Step 2:** Using the result $\Gamma$ from ILP analysis, the performance guarantees are checked in the simulation, i.e. whether a periodic phase could be found in the application schedule as in Figure 3.

**Step 3:** If the termination conditions (guarantees are found or maximum execution time) are met, it stops and outputs the results; otherwise, it increases the considered $t$ in ILP analysis by $\Delta_t$, and goes back to **Step 1**.

Apparently, only when the guarantees are met in Step 3, the output results are valid. As the initial value of $t$ and $\Delta_t$ are application model dependent, empirically, we run the synchronous simulation with some sufficient deadlock avoiding buffer $\Gamma$ first and choose both $t$ and $\Delta_t$ as the periodic phase length $L_{period}$.

We decompose the adaptation scenarios into two phases and build a buffer dimensioning exploration framework as the following:

**Phase I:** To find the minimal backlog $B_{min}$ requirement to sustain the stable transmission during the configuration, i.e. to minimize the following objective function (Eq. 4)

$$\min: B_{min,i} \tag{4}$$

subject to Constraint 1 - 7 (in Appendix).

**Phase II:** To find the minimal buffer sizes, which could accumulate backlog no less than $B_{min,i}$ in buffer $FIFO_{out}$ after $t_{interR,i}$, i.e. to minimize the following objective function (Eq. 5)

$$\min: \gamma_i + \gamma_{out}. \quad \gamma_i \geqslant B'_i(t),\ \gamma_{out} \geqslant B'_{out}(t), \forall t \in \mathbb{N}_0 \tag{5}$$

subject to Constraint 1 - 6 and Constraint 8 (in Appendix).

The minimal buffer sizes dimensioned by the exploration can guarantee the performance of the adaptive streaming applications.

In our implementation, we use the public domain tool *lp_solve* as the ILP solver, and use the ForSyDe synchronous library [8] for simulation purpose. A Python script is used to glue them together. The script generates the ILP model for *lp_solve*, retrieves the output, and invokes the synchronous simulation to check the throughput in both phases in our framework.

## 6. Case Study

In this section, we use the implementation of our methodology for several experiments on both the example application of Figure 1 and an industrial applications from Thales Communications. For different specifications of the adaptive process(es), e.g. $t_{C,i}$ and $t_{R,i}$ for the processing resource $pr_i$, the minimum buffer sizes of the application are dimensioned, while the application throughput requirement $\rho_{out}$ is guaranteed.

### 6.1 The example application

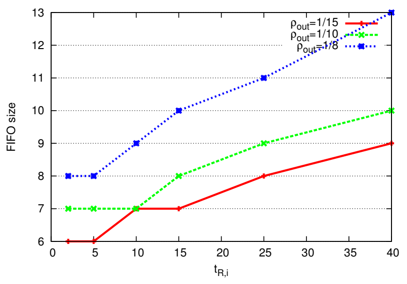
We assume the adaptive process $p_i$ in the example application has two different modes, and both configurations have the same properties (i.e., the same $m_i$, $n_i$, $t_{R,i}$, $t_{C,i}$, etc). We elaborate the model shown in Figure 1 with concrete numbers (i.e. $n_{in} = 2, m_i = 2, n_i = 3, m_{out} = 1$ and $t_1 = 1$) and assume the two succeeding reconfigurations of $p_1$ have a minimum interval $t_{interR,i} = 50$ slots.

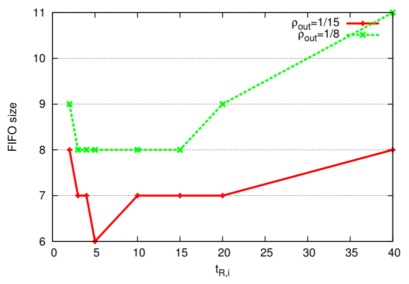
| **options** | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 |
|---|---|---|---|---|---|---|---|---|
| $t_{R,i}$ | 2 | 3 | 4 | 5 | 10 | 15 | 20 | 40 |
| $t_{C,i}$ | 16 | 13 | 12 | 10 | 8 | 6 | 5 | 4 |
| $A_M$ | 0.5 | 0.8 | 1.0 | 1.3 | 2.5 | 3.8 | 5.0 | 10 |

Table 1: Design options for the adaptive process $p_i$

First, we assume that different design options have varying reconfiguration time $t_{R,i}$, as listed out in the second row of Table 1, but have a fixed latency $t_{C,i}$ (i.e. $t_{C,i} = 10$), and evaluate the FIFO sizes requirement. Figure 5a shows the minimal FIFO sizes needed upon different $t_{R,i}$, corresponding to three output data rates $\rho_{out}$. To consider the two concerns $t_{R,i}$ and $\rho_{out}$ separately, apparently, higher $\rho_{out}$ demands larger FIFO sizes, so do the design options with higher $t_{R,i}$.

Instead, in the following scenario, we choose the design options according to the reconfiguration properties listed out in Table 5b, which conforms to $t_{R,i} = k_R A_{M,i}$ with $k_R = 10.0$ and an assumed relation $t_{C,i} \propto \lceil \frac{1}{\sqrt{t_{R,i}}} \rceil$. These design options show different implementation strategies in


(a) FIFO size upon varying $t_{R,i}$ & fixed $t_{C,i}$


(b) FIFO size upon varying $t_{R,i}$ & $t_{C,i}$

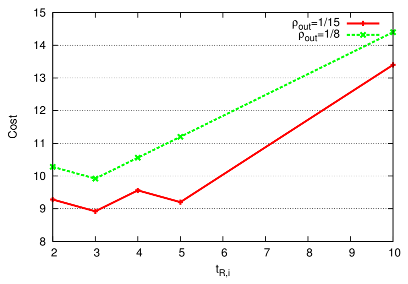
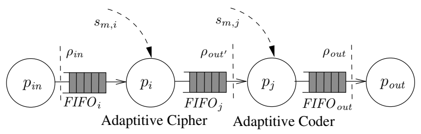
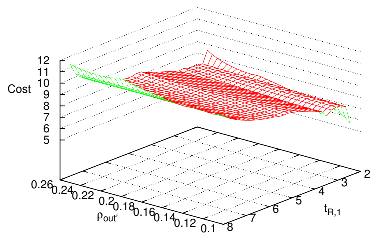

(c) Design cost upon varying $t_{R,i}$ & $t_{C,i}$

Figure 5: Experimental results of the example application



(a) Industrial application


(b) Design cost of JIT adaptation on the Cipher

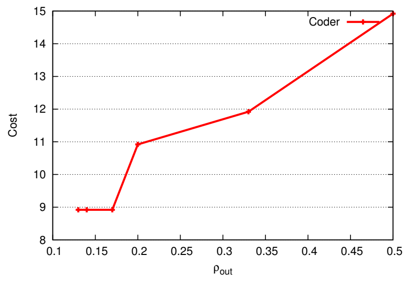

(c) Design cost of JIT adaptation on the Coder

Figure 6: Industrial application and experimental results

the speed and area trade-offs, e.g. an adder can be implemented as carry-lookahead adder (optimized for speed) or a ripple-adder (optimized for area).

Although, higher $\rho_{out}$ (i.e. $\rho_{out} = 1/8$) still demands larger FIFO sizes, the FIFO sizes are not monotonic to $t_{R,i}$ any more, as both $t_{C,i}$ and $t_{R,i}$ can affect the buffer requirement to sustain $\rho_{out}$ during reconfiguration. We can see that the design options with $t_{R,i}$ close to 5 need less buffer.

With a given compression ratio $k_C = 4.0$, the design costs are evaluated. As the design options after #5 simply show a fast monotonically increasing cost, we only present the cost of design option #1-5 for clarity in Figure 5c. We see higher throughput requirement still leads to larger design costs. However, the design costs heavily depend on the $t_{C,i}$ and $t_{R,i}$ trade-off, i.e. the speed and time trade-off, and #2 with $t_{R,i} = 3$ shows the minimum cost.

## 6.2 An industrial application

To evaluate the potential of our methodology in adaptive systems, we use it on an industrial application. We focus on the reconfigurable part of the application model as shown in Figure 6a, in which the specification parameters are omitted for clarity. There are two adaptive processes (modules): the Cipher $p_i$ and the Coder $p_j$. Each of them receives the adaptation control signal $s_{m,i}$ or $s_{m,j}$ from the environment, and can change the working modes among three possibilities (i.e. algorithm 1-3 for the Cipher, and BT/BL/BR coder for the Coder). The input/output constraints are captured by $\rho_{in}$ and $\rho_{out}$. The design objective is to minimize the buffer requirement to sustain the stable transmission with $\rho_{out}$, when either or both of the two modules are in configuration.

We decouple the two reconfigurable modules and solve them individually. The first one is to find the minimum buffer sizes for the Cipher buffers $FIFO_i$ and $FIFO_j$ to meet the throughput requirement $\rho_{out'}$, which is derived from the requirement from $\rho_{out}$. The second step is to find the minimum play-out buffer $FIFO_{out}$ for the Coder module, within the input constraints $\rho_{out'}$ and the output requirements $\rho_{out}$. Especially, for both the Cipher and Coder, all the reconfiguration possibilities (changing from one mode to the other, i.e. $3 \times (3-1) = 6$) are explored, and we choose the worst case buffer requirements to guarantee other adaptations.

The results of different implementation strategies $t_{R,i}$ are shown in Figure 6b and 6c. For the Cipher, the design costs are not monotonic to $t_{R,i}$. In both modules, the design cost increases with the throughput $\rho_{out'}$ or $\rho_{out}$.

It shows that our framework works for adaptive systems with a series of adaptive processes (modules) as well.

## 7. Conclusion

The experimental results show that our framework suits well to exploit the adaptivity of different design options, without losing efficiency, of real-time streaming applications. Especially, the industrial case study shows the capability of our methodology to cope with the sequential composition of adaptive systems.

In the future, we plan to extend our general approach on more implementation alternatives on adaptation strategies and target architectures.

## Appendix

### List of linear constraints

If not explicitly clarified in the linear constraints below, $\forall t \in \mathbb{N}_0$. We assume the designer will specify some initial values (e.g. $C_i(0) = 0$ in this paper), which are thus not constrained (considered) in the following semantics.

**Constraint 1** *For the processing resource $pr_i$, $R'_i[t]$ and $C_i[t]$ follow the static input/output tokens ratio as*

$$R'_i[t] \cdot m_i = C_i[t] \cdot n_i \tag{A-1}$$

**Constraint 2** *The incoming tokens to the processing resource $pr_i$ takes at least $t_{C,i}$ slots to be served.*

$$R_i(t) - C_i(t + \Delta_t) \geqslant 0, \quad 1 \leqslant \Delta_t \leqslant t_{C,i} \tag{A-2}$$

**Constraint 3** *For the processing resource $pr_i$, the demand function reserves vacant space $t_{C,i}$ slots in advance.*

$$D_i(t) = R_{out}(t + t_{C,i}) \tag{A-3}$$

**Constraint 4** *The buffer size $\gamma_i$ of the processing resource $pr_i$ satisfies the maximum buffer space requirement.*

$$\gamma_i \geqslant D_{in}(t) - C_i(t) \tag{A-4}$$

**Constraint 5** *The processing resource $pr_i$ have computation latency $t_{C,i}$ and input/output data tokens $m_i$ and $n_i$.*

$$C_i(t+t_{C,i})-C_i(t)=m_i\cdot\mathbf{k}_{C_i,t+t_{C,i}}, \quad \mathbf{k}_{C_i,t+t_{C,i}}\in\{0,1\} \tag{A-5}$$

$$D_i(t+t_{C,i})-D_i(t)=n_i\cdot\mathbf{k}_{D_i,t+t_{C,i}}, \quad \mathbf{k}_{D_i,t+t_{C,i}}\in\{0,1\} \tag{A-6}$$

**Constraint 6** *For the processing resource $pr_i$ with computation latency $t_{C,i}$, the arrival function differential does not exceed the peak data rate $\rho_{in}$.*

$$R_i(t + t_{C,i}) - R_i(t) \quad \leqslant \quad t_{C,i} \cdot \rho_{in} \tag{A-7}$$

**Constraint 7** *Especially, for the processing resource $pr_i$ using JIT adaptation after $t_{interR,i}$, the reconfiguration takes $t_{R,i}$ time (the processing capability stalls).*

$$C_i(t + t_{R,i}) - C_i(t) = 0, \quad t \geqslant t_{interR,i} \tag{A-8}$$

**Constraint 8** *Especially, for the processing resource $pr_i$ using JIT adaptation after $t_{interR,i}$, the backlog is required to have a minimum amount of $B_{min,i}$.*

$$B_i(t) \geqslant B_{min,i}, \quad t \geqslant t_{interR,i} \tag{A-9}$$

## Acknowledgments

This research has been partially supported by the ANDRES project within FP6, the sixth framework programme of the European Commission.

## References

[1] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. D. Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83, January 2003.

[2] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, LNCS, 2001.

[3] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, pages 190–195, Washington, DC, USA, 2003. IEEE Computer Society.

[4] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, 1995.

[5] R. Govindarajan, G. R. Gao, and P. Desai. Minimizing buffer requirements under rate-optimal schedule in regular dataflow networks. *Journal of VLSI Signal Processing*, 31(3):207–229, July 2002.

[6] S. Künzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 236–241, 2006.

[7] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36(1):24–35, January 1987.

[8] I. Sander and A. Jantsch. Modelling adaptive systems in ForSyDe. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 200(2):39–54, 2008. First Workshop on Verification of Adaptive Systems (VerAS 2007).

[9] S. Sriram and S. S. Bhattacharyya. *Embedded multiprocessors: Scheduling and synchronization*. CRC Press, 2000.

[10] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Design Automation Conference, DAC '06*, pages 899–904, San Francisco, California, USA, July 2006.