

A Flow Regulator for On-Chip Communication

Zhonghai Lu, Dimitris Brachos and Axel Jantsch
 KTH - The Royal Institute of Technology, Stockholm, Sweden
 Email: {zhonghai, brachos, axel}@kth.se

Abstract—We have proposed (σ, ρ) -based flow regulation as a design instrument for System-on-Chip (SoC) architects to control quality-of-service and achieve cost-effective communication, where σ bounds the traffic burstiness and ρ the traffic rate. In this paper, we present a hardware implementation of the regulator. We discuss its microarchitecture. Based on this microarchitecture, we design, implement and synthesize a multi-flow regulator for AXI. Our experiments show the effectiveness of such a regulation device on the control of delay, jitter and buffer requirements.

I. INTRODUCTION

IPs for a SoC are typically developed concurrently using a standard interface, for example, AXI [1] or OCP. Despite the standard interfaces, integrating IPs to a SoC infrastructure presents challenges because (1) traffic flows from IPs are diverse and typically have stringent performance constraints; (2) the impact of interferences among traffic flows is hard to analyze; (3) due to the cost and power constraint, buffers in the SoC infrastructure must not be over-dimensioned while still satisfying performance requirements even under worst case conditions.

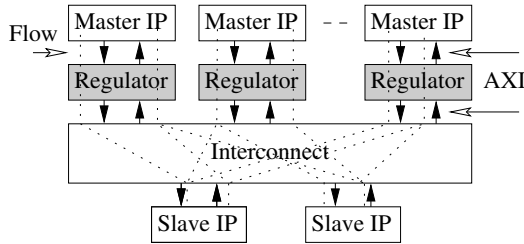


Fig. 1. IP Integration in SoCs

Figure 1 illustrates the approach that we have proposed and investigated in [2] for addressing the IP integration problem. Master IPs send read and write requests to slave IPs which respond with read data and write acknowledgements. The admission of traffic flows from master IPs into the SoC infrastructure can be controlled by a regulator rather than injecting them as soon as possible. In this way, we can control Quality-of-Service (QoS) and achieve cost-effective communication. To lay a solid foundation for our approach, our regulation has been based on *network calculus* [3][4][5][6]. By importing and extending the analytical methods from network calculus, we can obtain worst-case delay and backlog bounds.

The aim of this paper is to quantify the hardware speed and cost of the regulator and further show the benefits of using the regulator in controlling communication performance and buffer requirements. To this end, we present implementation microarchitecture of the regulator and a specific multi-flow regulator implementation for AXI, and design experiments on a 16-node network-on-chip (NoC) to show its benefits via analysis and simulation.

II. THE CONCEPTS OF FLOW REGULATION

A. Network Calculus Basics

In network calculus [3], a flow $F(t)$ represents the accumulated number of bits transferred in the time interval $[0, t]$. To capture the average and peak characteristics of a flow, it uses TSPEC (Traffic SPECification). With TSPEC, F is characterized by an *arrival curve* $\alpha(t) = \min(L + pt, \sigma + \rho t)$ in which L is the maximum transfer size, p the peak rate ($p \geq \rho$), σ the burstiness ($\sigma \geq L$), and ρ the average (sustainable) rate. We denote this $F \sim (L, p, \sigma, \rho)$.

Network calculus uses the abstraction of *service curve* to model a network element (node) processing traffic flows. A service curve reflects the processing latency and service capability of the node. A well-formulated service model is the latency-rate function $\beta_{R,T} = R(t - T)^+$, where R is the minimum service rate and T the maximum processing latency of the node [5]. Notation $x^+ = x$ if $x > 0$; $x^+ = 0$, otherwise.

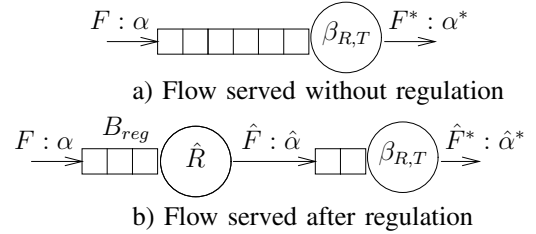


Fig. 2. Flow served by a latency-rate server

As depicted in Figure 2a, a TSPEC flow $F \sim (L, p, \sigma, \rho)$ (denoted as $F : \alpha$) is served by a node guaranteeing a latency-rate service $\beta_{R,T}$. According to [3], the maximum delay for the flow is bounded by Eq. (1) and the buffer required for the flow is bounded by Eq. (2):

$$\bar{D} = \frac{L + \theta(p - R)^+}{R} + T \quad (1)$$

$$\bar{B} = \sigma + \rho T + (\theta - T)^+ [(p - R)^+ - p + \rho] \quad (2)$$

where $\theta = (\sigma - L)/(p - \rho)$. The output flow F^* is bounded by another affine arrival curve $\alpha^*(t) = (\sigma + \rho T) + \rho t$, $\theta \leq T$; $\alpha^*(t) = \min\{(T+t)(\min(p, R)) + L + \theta(p - R)^+, (\sigma + \rho T) + \rho t\}$, $\theta > T$.

B. Regulation Spectrum

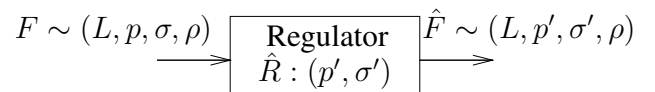


Fig. 3. Flow regulation.

TSPEC can be used to characterize flows. It can also be used to define a traffic regulator. Figure 3 shows that an input flow F is reshaped by a regulation component (regulator), resulting in an output flow \hat{F} . We assume that the regulator has the same input and output data unit, *transfer*, and the same input and output capacity C transfers/cycle. We also assume that F 's average bandwidth requirement must be preserved.

Given a flow F bounded by a TSPEC (L, p, σ, ρ) ($L \leq \sigma$, $\rho \leq p \leq C$), F can be losslessly reshaped by the regulator $\hat{R} : (p', \sigma')$ and the output flow \hat{F} is characterized by the four parameters (L, p', σ', ρ) , where $p' \in [\rho, p]$, $\sigma' \in [L, \sigma]$. "Lossless" means that \hat{F} has the same L and average rate ρ as F . However, regulation may change F 's peak rate and burstiness. The two intervals $p' \in [\rho, p]$ and $\sigma' \in [L, \sigma]$ are called the *regulation spectrum*, where the former is for the regulation of peak rate and the latter for the regulation of traffic burstiness.

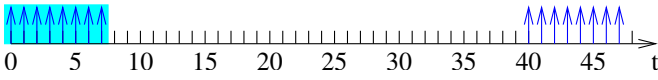


Fig. 4. Flow without regulation.

Let us consider a periodic data flow F that generates one transaction of 8 transfers every 40 cycles, as shown in Figure 4. Thus its average rate $\rho = 8/40 = 0.2$ transfer/cycle and burstiness $\sigma = 8 - 0.2 \cdot (8 - 1) = 6.6$ transfers. Its peak rate p is 1 transfer/cycle. Its TSPEC can therefore be written as $F \sim (1, 1, 6.6, 0.2)$. As can be seen, F has a maximal burstiness since all transfers in one period are launched one right after the other.

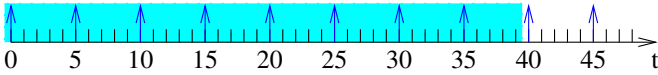


Fig. 5. Flow with strong regulation.

If the admission of flow F is controlled by regulator $\hat{R} : (0.2, 1)$, F is re-shaped as shown in Figure 5 and \hat{F} has a TSPEC, $\hat{F} \sim (1, 0.2, 1, 0.2)$. This is the other extreme case where the burstiness ($\sigma = L = 1$) is minimal and its peak rate equals average rate ($p = \rho = 0.2$). As a result, the 8 transfers are evenly distributed over the period.

The regulation spectrum defines the upper and lower limits of regulation. Figure 2b shows how the flow is served after regulation.

C. Regulation Mechanism

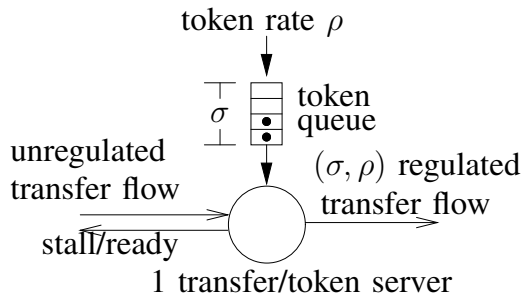


Fig. 6. (σ, ρ) -based Regulation Mechanism

The regulator may be implemented using the token-bucket mechanism [7] as shown in Figure 6. The token queue has a size

of σ . Initially the token queue is full. The 1-transfer/token server admits one transfer by de-asserting the "stall" signal as long as the token queue is not empty. The token queue is realized by a saturating credit counter that increments at rate ρ and saturates when it reaches a count of σ . A transfer can be transmitted if and only if the credit counter is positive (at least one token available). Each time a transfer is sent, the counter is decremented by 1.

III. REGULATOR IMPLEMENTATION

A. Microarchitecture

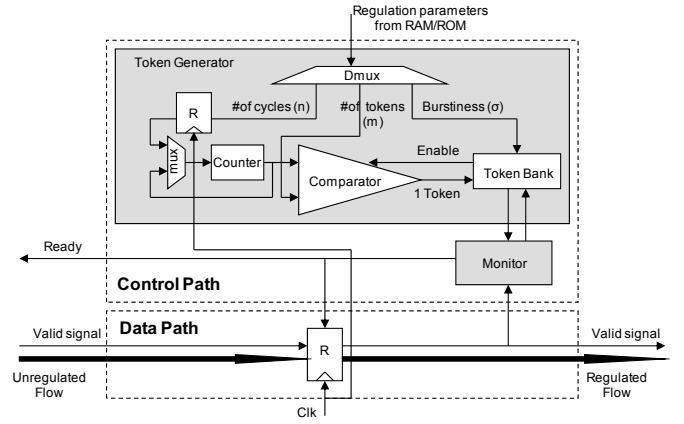


Fig. 7. Regulator implementation in hardware.

Figure 7 shows the micro-architecture of the regulator, according to the token-based regulation mechanism. The regulator consists of a control path and a data path. The simple data path is for the flow traversal under the regulation of the control path. The control path governs whether the master is allowed to launch a transfer or not by asserting the "Ready" signal. It also monitors the outgoing transfer flow to check if there is a transfer sent out. The control path has two parts: a token generator and a flow monitor. The token generator reads three regulation parameters, namely, number of cycles (n), number of tokens (m) and burstiness (σ) from a storage, for example, RAM or ROM. The regulated flow rate ρ is derived from n and m by $\rho = m/n$. After loading the parameters, the token bank has an initial value of 1 or σ (if $\sigma > 1$). This is also the maximum size of the token bank. To facilitate the description, let x be the number of real-time tokens in the token bank. Initially, $x = \sigma$. The regulator asserts the "Ready" signal. If there is a transfer sent out, x decrements by 1, i.e., $x = x - 1$. The comparator is responsible for filling the token bank. When $x = \sigma$, it is disabled; enabled, otherwise. The counter is a re-settable count-down counter. It has an initial value of n . It counts down at each cycle. After reaching 1, the counter is re-set to n . The comparator compares the value from the counter, c , and $n - m + 1$. If $c \leq n - m + 1$, one token is generated; otherwise, no token is created. In this way, m tokens will be created each and every n cycles, thus ensuring the rate of $\rho (= m/n)$. In hardware, n, m, σ and other internal data types are all implemented as an integer. This sacrifices precision but saves area in silicon.

B. AXI-Based Regulators

We implemented master and slave regulators at the transfer level for AXI [1]. This means that σ and ρ are counted in *transfers* and *transfer/cycle*, respectively. AXI features five wide

parallel channels: AW (write address), W (write data), B (write response), AR (read address), and B (read data). We can place a regulator for each channel, if necessary. In the following we exemplify a master regulator in detail.

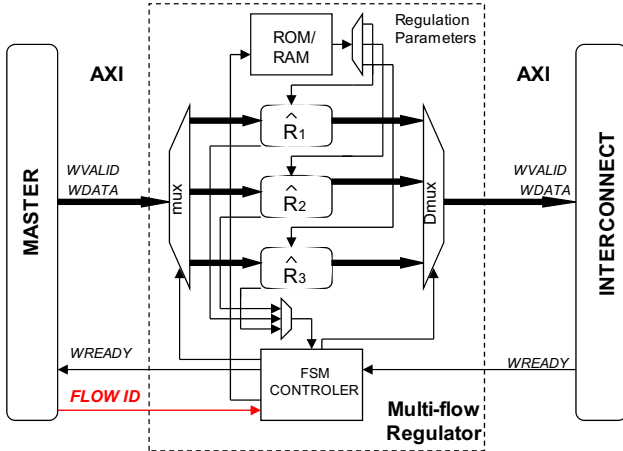


Fig. 8. A multiflow regulator for AXI.

Figure 8 shows a master regulator for the write data. The regulator, which is placed on the W channel, is capable of regulating multiple dynamic flows concurrently, thus called *multiflow regulator*. To deal with multiple flows, we can use one shared regulator. But to enhance performance, we can place multiple regulators. In this figure, it shows three parallel regulators \hat{R}_1 , \hat{R}_2 and \hat{R}_3 . Each regulator has a microarchitecture as shown in Figure 7. The RAM/ROM stores flow regulation parameters. Each entry inside is a pair with four parameters $(f_i, (n_i, m_i, \sigma_i))$, where f_i is Flow ID (identifier) and (n_i, m_i, σ_i) regulation parameters for flow f_i . We use the WREADY signal to act as the Ready signal in Figure 7. The additional signal that we add to AXI is the Flow ID, which is used to distinguish one flow from others. We say a flow *active* if its regulation parameters are already loaded into a regulator. Otherwise, the flow is *inactive*.

The multiflow regulator first identifies the flow ID. Then, if the flow is active, it conducts regulation accordingly and immediately, taking one cycle; if the flow is inactive, it fetches the regulation parameters associated with this particular flow from the RAM/ROM table and loads these parameters to an available regulator or reconfigure the recently-not-used regulator. This reconfiguration takes two extra cycles (one cycle for reading parameters and the other for configuring the regulator). The coordination is performed by the FSM controller.

Slave regulators are similar to the master regulator with slight difference in flow ID management since a slave does not generate a flow ID. Therefore, the slave regulator has to record the flow ID together with its AXI transaction ID, and later use the AXI transaction ID to retrieve its flow ID.

C. Synthesis Results

To quantify area and speed, we synthesized the AXI multiflow regulator (Figure 8) with Synopsys tools using 180 nm technology. The counter is set to 8 bits, and n , m and p take 10 bits each. The parameter table has three entries, and are synthesized into registers. When optimized for area, the regulator consumes 5K gates and the speed is 730 MHz; When optimized for timing, the gate count is 7K and speed 860 MHz.

IV. EXPERIMENTS

A. Experiment Purpose

The purpose of the experiments is to verify the correct function of the regulator and to highlight the impact of regulation on communication performance (delay and jitter) and buffer requirements in the communication infrastructure.

B. Experimental Setup

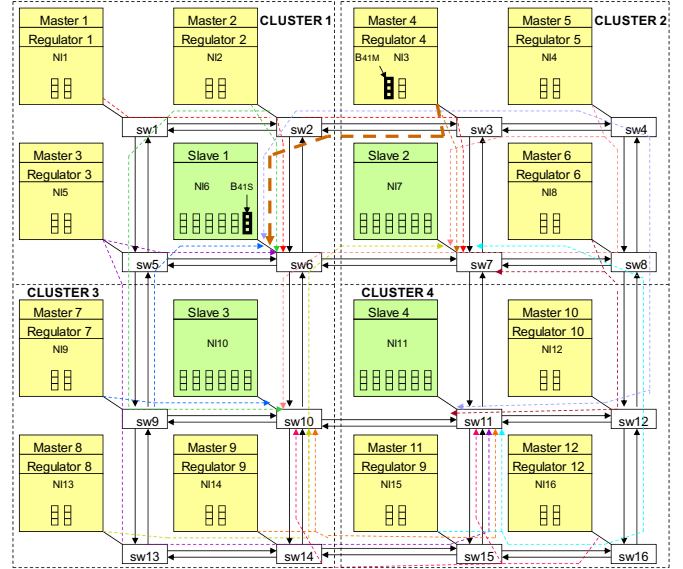


Fig. 9. Experimental architecture

1) *Network*: As illustrated in Figure 9, the experimental architecture is a 4×4 mesh synchronous network. It has 12 master nodes and 4 slave nodes. Each master is equipped with a regulator (Figure 8) and a network interface (NI). The NIs encapsulate transfers into packets before sending them to the network and de-encapsulate packets back into transfers after receiving them from the network. The 4 slaves are located in the four center nodes of the mesh in order to reduce overall communication distance. The network is partitioned into 4 clusters. Each cluster contains 3 master nodes and 1 slave node. Specifically, Cluster i ($i = 1, 2, 3, 4$) hosts 3 master nodes $i, i+1, i+2$ and 1 slave node i . The master NIs use per-flow buffers to store the packets before they are admitted into the network. The slave NIs, which have one buffer per flow, multiplex transactions to the slaves. The arbitration policy is round-robin. We assume that the slaves can process multiple transactions concurrently and they respond to transactions with 1 cycle delay, as a typical SRAM does.

To investigate the regulation effect, we have chosen to use a contention-free virtual circuit (VC) network [8]. Each communication master-slave pair has a time-division-multiplexing (TDM) VC established. The reasons for using VC rather than a best-effort service are (1) using the VC service can avoid stochastic behavior of best-effort services; (2) as the VC service itself has stronger regulation effect on traffic than the BE service, we show that regulation makes sense even for strongly-regulated services.

2) *Traffic*: In our experiments, we use write transactions and focus on the regulation of write data transactions. The write responses from the slave are not regulated since they contain only one transfer.

We generate two layers of unicast communication: *intra-cluster communication* and *inter-cluster communication*. For the intra-cluster communication, all 3 masters can communicate with the slave within each cluster. For example, in Cluster 1, all 3 masters 1, 2 and 3 communicate with the slave 1. For the inter-cluster communication, one master in each cluster is responsible for communicating with the slave in another cluster. For instance, for Cluster 1, master 1, 2 and 3 communicates with the slave 2 in Cluster 2, slave 3 in Cluster 3 and slave 4 in Cluster 4, respectively. In this way, each master is able to communicate with each slave directly or indirectly. This symmetric organization can also facilitate workload balancing, avoiding hot spots. According to the above communication pattern, each master has 2 corresponding slaves (one inside its cluster and the other outside) and each slave has 6 corresponding masters. Altogether we have 24 communicating source-destination pairs, thus 24 flows. Each master allows up to 2 outstanding transactions.

The rate of the flows is set in the range $[0.0078, 0.0625]$. The injection window (period) is 256 cycles for all flows. A rate of 0.0625 transfers/cycle means that 16 transfers are injected on average every 256 cycles (16/256). The size of transactions of the flows varies from 2 to 16 transfers, resulting in burstiness in the range of $[1.99, 15.06]$ transfers. The transfers of a transaction are launched one after the other consecutively in the beginning of the period if no regulation is applied. While in simulations, we apply regulation to all flows in the same way, i.e., all strong or medium regulation. Also, masters are stalled when they are not allowed to send requests in the case of regulation.

C. Network Calculus Analysis

In order to derive per-flow theoretical delay and buffer bounds, we build network calculus analysis models for the experiments following [2]. Specifically, we model VCs and the round-robin multiplexors as a latency-rate server. A VC is modeled as $\beta_{Bw,T}$, where Bw is the VC service bandwidth and T processing delay. The round-robin multiplexors are modeled as $\beta_{0.16,5}$, since they offer 0.16 transfers/cycle for each flow and the maximum delay to serve a flow is 5 cycles.

All flows can be characterized with TSPEC and denoted as $F_{(src,dst)} \sim (L, p, \sigma, \rho)$, where src and dst are the flow's source and destination, respectively. We apply different regulation strength on the flows: *Unregulation*, *Medium-regulation* and *Strong regulation*. The unregulated case means there is no regulation effect. The regulator virtually exists and the flow keeps its maximum burstiness σ . The strong regulation is the other extreme with the highest regulation, resulting in burstiness of 1. The medium regulation situates in between, resulting in burstiness in the range $(1, \sigma)$.

To tighten the delay bound, we used the concatenated node model [3]. To tighten the buffer bound, we used the revised output arrival curve in [2].

D. Analysis and Simulation Results

We show both analysis and simulation results for delay in cycles and buffer requirements in transfers. In the experiments, flow $F_{(4,1)}$ passing through Switch 3, 2 and 6 experiences the worst-case transaction delay. We report detailed results for this flow in Table I. In the Table, \bar{D} and \bar{B} are analyzed transfer delay bound and network buffer bound, respectively. D_{max} , D_{avg} and D_{min} are simulated maximum, average and minimum transfer

delay. Note that the delays are counted from when a request is sent until it reaches the slave. B is the required number of buffers in simulation, referring to the sum of B_{41M} and B_{41S} in Figure 9. For the three regulation cases, Unregulation keeps $F_{(4,1)}$'s TSPEC, $\hat{F}_{(4,1)} = F_{(4,1)} \sim (1, 1, 13.27, 0.054)$; Medium Regulation shapes $F_{(4,1)}$ into $\hat{F}_{(4,1)} \sim (1, 1, 6.67, 0.054)$; Strong Regulation shapes $F_{(4,1)}$ into $\hat{F}_{(4,1)} \sim (1, 0.054, 1, 0.05)$.

	Delay and Jitter (cycles)					Backlog	
	\bar{D}	D_{max}	D_{avg}	D_{min}	J	\bar{B}	B
Unreg.	129	117	58.8	13	104	26.6	26
Medium Reg.	71.7	61	37	13	48	14.1	12
Strong Reg.	22.2	13	13	13	0	2.7	2

TABLE I
ANALYZED VS. SIMULATED TRANSFER DELAY AND BACKLOG

From Table I, we can observe (1) Analyzed delay and buffer bounds (\bar{D} and \bar{B}) envelop the worst-case delay (D_{max}) and buffer requirement (B); (2) Regulation greatly reduces jitter J calculated by $J = D_{max} - D_{min}$. We can see that, as the regulation strength increases, J is reduced from 104 to 48 (Medium Reg.) to 0 (Strong Reg.) cycles; (3) Regulation significantly decreases the network buffer requirement (\bar{B}) from 26.6 to 14.1 (Medium Reg.) to 2.7 (Strong Reg.) transfers.

V. CONCLUSION

IP integration requires the provision of performance guarantees for traffic flows and efficient buffer dimensioning techniques. In this paper, based on the concepts of formal regulation, we have presented the hardware micro-architecture and implementation of a multiflow regulator. This regulator can be deployed as a modular device between an IP and the interconnect. Synthesis results show that the regulator for AXI can run up to about 800 MHz with a gate count of 7K. The regulation changes the burstiness and timing of traffic flows, and can be used to control delay, jitter and reduce buffer requirements in the SoC infrastructure. We should also mention that, since regulation holds back traffic, flows should not be over-regulated in order to satisfy the performance constraints.

In the future, we will analyze the gain of regulation in power consumption, especially for leakage power. The potential is high because it can significantly reduce network backlogs.

REFERENCES

- [1] ARM, "AMBA Advanced eXtensible Interface (AXI) protocol specification, Version 1.0," <http://www.amba.com>.
- [2] Z. Lu, M. Millberg, A. Jantsch, A. Bruce, P. van der Wolf and T. Henriksson, "Flow Regulation for On-Chip Communication", *Proceedings of the 2009 Design, Automation and Test in Europe Conference (DATE'09)*, Nice, France, April 2009.
- [3] J.-Y. L. Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Number 2050 in LNCS, 2004.
- [4] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation; part II: Network analysis," *IEEE Transactions on Information Theory*, vol. 37, no. 1, January 1991.
- [5] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 611–624, October 1998.
- [6] C. Chang, *Performance Guarantees in Communication Networks*. Springer-Verlag, 2000.
- [7] P. P. Tang and T. Yuan Charles Tai, "Network traffic characterization using token bucket model," in *Proceedings of the IEEE INFOCOM '99*, March 1999, pp. 51–62.
- [8] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1021–1034, 2008.