Network Calculus Applied to Verification of Memory Access Performance in SoCs

Tomas Henriksson and Pieter van der Wolf
NXP Semiconductors Research
Eindhoven, The Netherlands
tomas.henriksson@nxp.com

Axel Jantsch
Royal Institute of Technology
Stockholm, Sweden

Alistair Bruce
ARM
Sheffield, U.K.

Abstract

SoCs for multimedia applications typically use only one port to off-chip DRAM for cost reasons. The sharing of interconnect and the off-chip DRAM port by several IP blocks makes the performance of a SoC under design hard to predict. Network calculus defines the concept of flow and has been successfully used to analyse the performance of communication networks. We propose to apply network calculus to the verification of memory access latencies. Two novel network elements, packet stretcher and packet compressor, are used to model the SoC interconnect and DRAM controller. We further extend the flow concept with a degree and make use of the peak characteristics of a flow to tighten the bounds in the analysis. We present a video playback case study and show that the proposed application of network calculus allows us to statically verify that all requirements on memory access latency are fulfilled.

1. Introduction

System-on-Chips (SoCs) for multimedia applications typically contain programmable as well as function-specific IP blocks for a good trade-off between flexibility and efficiency. For many video functions large data sets, e.g. multiple video frames, have to be stored. For cost reasons SoCs typically use off-chip DRAM to store these data sets. Further, cost constraints force SoC designers to use a single DRAM port per SoC. The memory accesses from the IP blocks to the shared DRAM need to be arbitrated in the interconnect and memory interface before they reach the DRAM. One of the main challenges in SoC design is to assure that the latency of the memory accesses is below certain upper bounds, in order to meet the deadlines of the tasks that execute on the IP blocks. Specifically it needs to be verified that deadlines are met under all circumstances, e.g. for all video content and for all possible interactions in the interconnect and memory controller. Therefore a worst-case analysis is required.

We propose a method for static analysis of worst-case memory access performance based on network calculus theory [1]. In this method the performance requirements of the IP blocks are captured as a set of traffic flows with associated latency constraints. The interconnect and memory architecture is modeled as a set of interconnected network elements. We verify for the specified traffic flows whether the worst-case delays incurred by the interconnect and memory architecture satisfy the latency constraints associated with these traffic flows. This paper focuses on the most critical part: the accesses to DRAM.

This work was partly funded by the European IST-2004-27580 SPRINT project.

The aim of this paper is to show that network calculus can be applied to model SoCs and DRAM for the analysis of memory access performance. We present a model of a DRAM with its associated controller. The model is based on two novel network elements, packet stretcher and packet compressor. We further extend the flow concept of network calculus with a degree and make use of the peak characteristics of a flow to tighten the bounds in the analysis.

In Section 2 the performance analysis problem is described. In Section 3 related work is discussed. In Section 4, network calculus is introduced. In Section 5 the application of network calculus to SoC and DRAM modelling is presented. In Section 6 techniques to tighten the bounds in the analysis are explained. In Section 7 a case study for a video playback application is presented. Section 8 concludes.

2. Problem statement

The general problem we address is how to statically verify that a SoC infrastructure, i.e. an interconnect and memory architecture, can serve the communication requirements of the IP blocks that implement application tasks. Starting point is that we can characterize the communication requirements of the IP blocks by means of a set of traffic flows and associated latency constraints. The traffic flows specify the upper bounds for the traffic from/to the IP blocks. We assume that it has been validated for each IP block that if the SoC infrastructure satisfies the specified latency constraints for the specified traffic flows then the IP block will meet the required performance for executing its task(s).

Analysis of the performance of accesses to off-chip DRAM is particularly challenging because DRAM access latencies can vary significantly due to arbitration in the DRAM controller and DRAM access characteristics. The specific problem we address in this paper is to derive worst-case bounds on latencies for DRAM accesses where these bounds are sufficiently tight to make the analysis useful in practical SoC design cases. The traffic flows should be able to capture periodic traffic, as typically generated by dedicated hardware blocks, as well as bursty traffic from programmable processors. Traffic flows can be read / write request flows or read / write response flows. Transactions link responses to requests. Most IP blocks have only a small number of outstanding transactions, issuing new requests only after responses for previous requests have been received. The techniques for modelling and analysing the SoC infrastructure must be able to closely represent the processing of such flows and allow us to derive tight bounds on the latencies of the transactions.

3. Related work

In the context of communication networks a wealth of theory has been developed to analyse network performance. Cruz has pioneered network calculus [1], which is a mathematical framework to derive worst-case bounds on latency, backlog and throughput. Based on Cruz' foundation Stiliadis and Varma [3] have developed a theory of Latency-Rate Servers. Latency-rate servers are an abstraction of network elements like delay elements, multiplexers and schedulers.

LeBoudec [2] has further developed the network calculus theory and based it on Min-Plus algebra. The basic elements in this algebra are arrival curves and service curves. Arrival curves bound traffic flows. Service curves describe output guarantees of network elements. Using arrival curves and service curves, the maximum backlog and maximum delay can be determined. LeBoudec has applied this theory to ATM and integrated service networks. Network calculus has also been applied outside communication networks. E.g. Thiele et al. have applied network calculus to hard real-time embedded systems [4].

SoC performance verification has been addressed by e.g. event streams in [5]. The focus is on task interaction and scheduling and not on the derivation of execution times of tasks, which is what we address. Deriving worst case execution times (WCETs) of tasks has been addressed by e.g. [6]. The memory access latencies are however assumed to be known. Staschulat [7] suggests to use the techniques from [5] in the SoC infrastructure in order to improve the modelling of memory accesses. Our work has a similar objective, but in addition focuses on correctly modelling the characteristics of DRAMs, including refresh cycles, bank conflicts and read-write turn around times.

4. Network calculus basics

Network calculus is based on bounds of traffic flows. If a flow is bounded by a monotonously non-decreasing function $b(\Delta t)$ then the amount of traffic that flows through a given point in the network during any period of length Δt is smaller than or equal to $b(\Delta t)$. A useful family of functions for concise descriptions of bounds of traffic flows are so called (σ, ρ) functions, where ρ is the average bandwidth and σ limits the burstiness of the flow. They have the form

$$b(\Delta t) = \sigma + \rho\,\Delta t, \qquad (1)$$

as illustrated in Figure 1.

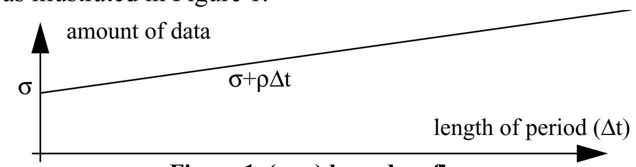

Figure 1. (σ, ρ) bound on flow

A flow has a constant packet size L and the flow travels on a link with a capacity C. There is a lower limit for the burstiness because of the packet size: $\sigma \geq L\,(1 - \rho/C)$. Based on the characterization of flows and the link capacity, Cruz has derived formulas to compute upper bounds for worst-case delay, average delay and backlog for a variety of network elements such as constant delay line, regulator, and a number of different multiplexers. The definition of delay is the time from the first bit of a packet enters a network element until the first bit of the packet exits the network element.

Multiplexers are the most interesting and challenging network elements. A busy period is defined to be a maximal interval of time such that data flows on the output link of the multiplexer at rate C_{out} throughout the interval. According to [1] no bit ever has to wait longer than the longest busy period in a multiplexer. With additional knowledge, the delay bound can be made tighter, e.g. a locally first-come-first-served (LFCFS) 2-input multiplexer with $C_1=C_2=C_{out}=C$ has the worst-case delay (Equation 4.27 in [1]):

$$D_1 \leq \frac{\sigma_2}{C-\rho_2} + \left(\frac{\sigma_1}{C-\rho_1}\right)\left(\frac{\rho_2}{C-\rho_2}\right) \qquad (2)$$

So far the network calculus theory has not been applied to analysis of memory access performance of SoCs. There are a number of characteristics in SoC architectures that are not easily taken into account by the network calculus theory.

- Packets change size while travelling through an on-chip network. For instance a read-request packet is transformed into a burst of data by the memory.
- Most network calculus formulas assume that network elements process data when there are packets on the inputs. DRAMs however spend time on preparation and refresh activities although there are requests available.
- The number of uncompleted transactions at a given time is often limited and can be statically determined in a SoC. This information can be used to derive significantly more realistic bounds on latency and buffer size. Network calculus does not take this into account.
- SoC flows have specific latency constraints that can be exploited. Often, the delay of an individual transaction is less interesting than the worst-case aggregate delay of a number of transactions; e.g. in video processing the deadlines are attached to processing of an entire frame and not to individual read or write transactions.

These specific characteristics make it worthwhile to adapt network calculus for the SoC domain in order to derive tighter bounds and avoid over-design of hardware resources.

5. SoC infrastructure model

Our contribution to the modelling of the SoC infrastructure consists of two parts, two new network elements and a method to model a DRAM by making use of these two new network elements.

5.1 Packet stretcher and packet compressor

We introduce two novel network elements, packet stretcher and packet compressor. These are used to model the request to response transformations in memory transactions and the clock cycles that are spent on DRAM preparation.

A packet stretcher changes the size of a packet from L_{in} to L_{out}, where $L_{out} > L_{in}$. The input link and the output link may have different capacities C_{in} and C_{out}. The bandwidth changes according to the packet stretching: $\rho_{out} = (L_{out}/L_{in})\ \rho_{in}$. To compute the burstiness σ_{out} we need one definition. The earliest possible arrival time of packet p in a burst that starts with packet 1 at time 0 is:

$$a(p) = \max\{(p\ L_{in})/C_{in,}\ ((p\ L_{in})-\sigma_{in})/\rho_{in}\} - L_{in}/C_{in} \quad (3)$$

The burstiness of the output of the packet stretcher is:

$$\sigma_{out} = \max_p\{p\ L_{out}-\rho_{out}(a(p)+L_{out}/C_{out})\} \quad (4)$$

It holds that $\sigma_{out} \leq (L_{out}/L_{in})\ \sigma_{in}$. This is explained by the fact that the time for a packet after the stretcher is longer than the time for a packet before the stretcher, $L_{in}/C_{in} \leq L_{out}/C_{out}$.

The delay of a packet stretcher is zero because the first bit of the output packet is sent at the same time as the first bit of the input packet arrives. The input packet inter-arrival time must be at least as long as the time it takes to send an output packet (L_{out}/C_{out}). If the input packet inter-arrival time could be shorter, a regulator is needed in conjunction with the packet stretcher in the model, see Figure 2. A regulator in conjunction with a packet stretcher has $\rho_q = (C_{out}/L_{out})\ L_{in}$ B/s and $\sigma_q = L_{in}(1-(C_{out}\ L_{in})/(L_{out}\ C_{in}))$ B. We have derived these characteristics directly from the fact that stretched packets should not overlap. We have further derived the maximum delay of the regulator as:

$$D = (C_{in}\ \sigma_{in}/(C_{in}-\rho_{in})-\sigma_q)/\rho_q - \sigma_{in}/(C_{in}-\rho_{in}) \quad (5)$$

This bound can be made tighter if the packet sizes are taken into account in a similar way as explained for multiplexers in Section 6.1.

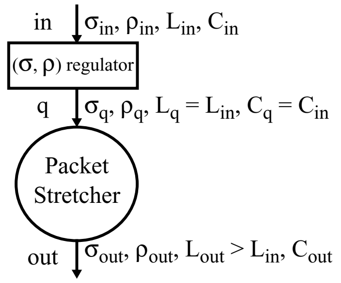
in
$\sigma_{in}, \rho_{in}, L_{in}, C_{in}$
(σ, ρ) regulator
q
$\sigma_q, \rho_q, L_q = L_{in}, C_q = C_{in}$
Packet Stretcher
out
$\sigma_{out}, \rho_{out}, L_{out} > L_{in}, C_{out}$

Figure 2. Regulator in conjunction with packet stretcher

A packet compressor changes the size from L_{in} to L_{out}, where $L_{out} < L_{in}$. The input link and the output link may have different capacities C_{in} and C_{out}. The delay of a packet compressor is 0. The output bandwidth $\rho_{out} = (L_{out}/L_{in})\ \rho_{in}$. The burstiness $\sigma_{out} = (L_{out}/L_{in})\ \sigma_{in} + (L_{in}/C_{in} - L_{out}/C_{out})\ \rho_{out}$.

5.2 DRAM model

The DRAM and the DRAM controller are modelled together. The model is depicted in Figure 3 for two flows. We model the multiplexing between requests as multiplexing between *abstract memory packets (AMPs).* The AMPs have a size that corresponds to how much of the raw capacity of the DRAM interface they use in the worst case. An AMP accounts for the time the memory controller is occupied with the transaction. The size of the AMPs depends on the size and alignment of transactions as well as the mapping of transactions to DRAM banks and the DRAM specification (DDR, DDR2, LPDDR, DDR3). If only long, well-aligned bursts that cyclically access all banks are used, the time for activating and precharging DRAM rows can be overlapped with data transfers and the AMPs do not include that time.

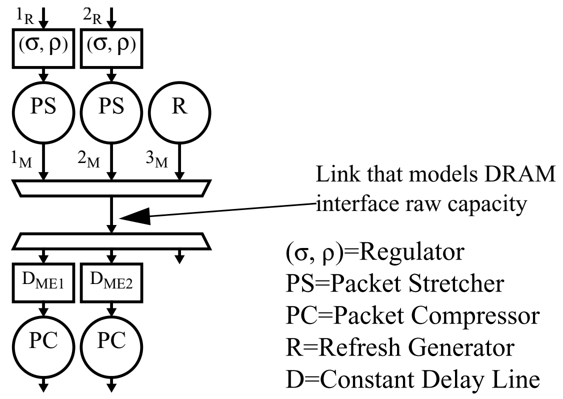

Figure 3. DRAM model

We assume that the write data is available when a corresponding write request is chosen by the multiplexer in the DRAM controller. In this way, the write data does not need to be modeled separately in the DRAM model.

The request flows (1_R and 2_R) are regulated and stretched into AMP flows (1_M and 2_M) on links with capacity equal to the raw DRAM interface capacity. The delay of the regulators in conjunction with the packet stretchers is denoted by D_{MR}. The DRAM is modelled as a link with the raw capacity of the DRAM interface. The multiplexer thus has the same capacity on all input/output links. The delay in the multiplexer in the DRAM controller is denoted by D_{MM}. An additional flow of AMPs is introduced to model the refresh cycles of the DRAM (3_M). In the model the refresh generator is a periodic source of AMPs. Some jitter is allowed on the refresh, so we assume that the refresh requests are multiplexed along with the normal requests before issued to the DRAM.

The multiplexer may use any arbitration policy. For example round-robin (RR) or fixed priority (FP) can be used. Equation (2) can be used to get an upper bound on D_{MM} for those policies, but tighter bounds can be computed by making use of more information as shown in the next section.

The AMPs are demultiplexed in the model to separate the response flows. The memory execution delay (D_{ME}) is then modelled as a constant delay line with the delay equal to the worst-case delay of the DRAM. If the master waits for the first bits of the response packet (e.g. critical-word-first cache line refill), the delay to the first bits of the packet is modeled. If the master waits for the complete response packet the delay to the last bits of the packet is modeled. A packet compressor converts AMPs to response packets.

The total delay from the time when a request enters the DRAM controller until the response is available is:

$$D_M=D_{MR}+D_{MM}+D_{ME} \quad (6)$$

In the actual hardware, no request regulators, packet stretchers, AMPs, or packet compressors are present. They are only present in the model. In the actual hardware, the requests are queued in the DRAM controller.

6. Tight bounds considerations

The modelling approach presented in the previous section makes it possible to compute latency bounds of SoC memory accesses. For SoC performance analysis it is however necessary to derive *tight* bounds. In this section we discuss how to tighten the bounds compared to network calculus theory by Cruz. This is done by a different analysis approach and by more accurate traffic descriptions.

6.1 Packet size and specific arbitration policy

Equation (2) is valid for all LFCFS arbitration policies. Also the packet size is not included in the formula. It provides an explicit and concise formula, but does not give tight bounds. We propose to make use of an iterative numerical method to derive the worst-case delays in a multiplexer. For the longest busy period (LBP) we compute the *earliest arrival time* of each packet according to Equation (3). Then the following procedure is repeated for every flow. We compute the *latest possible service time* of each packet according to the specific arbitration policy, the worst-case state of the arbiter, and the service times of previously served packets in the LBP. The delay of each packet in the flow is obtained by subtracting the *earliest arrival time* from the *latest possible service time*. A maximum operation over the delays of all packets in the flow in the LBP gives the worst-case delay for the flow.

6.2 Degree

We introduce the degree of a flow as the maximum number of packets of a flow that are in flight at any point in time. This is used to limit the maximum backlog and to get tighter bounds on the total delay for consecutive transactions. A degree of 1 implies that a packet never has to wait for another packet of the same flow. This also means that the backlog in any network element is no more than 1 packet.

The degree leads to later arrival times of packets because there is a dependency on the service time of previous packets. Equation (3) is not applicable when the degree is taken into account. The arrival times are in this case computed with an iterative numerical method similar to the one explained in Section 6.1 for service times.

6.3 Consecutive transactions

So far we have only discussed the analysis of the worst-case delay for a single transaction. In SoCs the worst-case aggregate delay of consecutive transactions is important, especially for the read flows of programmable processors. The latency constraint is specified as the maximum aggregate delay for all transactions in a period Δt. The straightforward way is to multiply the number of transactions with the maximum delay per transaction. One way to tighten the bounds for a multiplexer is to make use of the degree. The read flows of a programmable processor typically have a low degree. For a flow with degree 1 we know that maximally one packet of that flow waits for any packet of another flow in the case of RR arbitration. The number of packets from each flow during Δt is limited by $n_i(\Delta t) = \mathrm{ceil}((\sigma_i+\rho_i\Delta t)/L_i)$. Consider N flows that are ordered in increasing n_i. With RR arbitration we get for flow k (with degree 1) that the worst case aggregate delay of the multiplexer is bounded by:

$$D_{k,\,aggregate} \le \sum_{i=1}^{k-1} \langle n_i \frac{L_i}{C} \rangle + n_k \sum_{i=k}^{N} \frac{L_i}{C} \qquad (7)$$

6.4 Peak characteristics

We introduce peak characteristics of flows. Programmable processors exhibit flows with bursty characteristics. When such a flow is modelled with σ and ρ, σ is large. This leads to long worst-case delays in many network elements. By introducing peak characteristics, σ_p and ρ_p, a more accurate bound on the traffic can be described. During a burst, the average bandwidth is ρ_p, which is always larger than the long-term average bandwidth ρ, but smaller than the link capacity C. The burstiness during a burst, σ_p, is thereby smaller than the long-term burstiness σ. Typically ρ_p and σ_p are chosen so that $\sigma_p = L(1-\rho_p/C)$, the minimum possible burstiness.

The amount of data in a flow during any period of time of length Δt is bounded according to Equation (1). If we know C the bound can be restricted to $b(\Delta t) = \min\{C\Delta t, \sigma+\rho\Delta t\}$. With the peak characteristics (σ_p, ρ_p) it can be further restricted, $b(\Delta t) = \min\{C\Delta t, \sigma+\rho\Delta t, \sigma_p+\rho_p\Delta t\}$, see Figure 4. The bound $b(\Delta t)$ is the thick line. This can be generalized to any number of (σ_i, ρ_i) pairs to describe the traffic. If more (σ_i, ρ_i) pairs are used, tighter bounds can be derived, but the analysis becomes more cumbersome. For the arrival time calculation of every packet all (σ_i, ρ_i) pairs have to be checked.

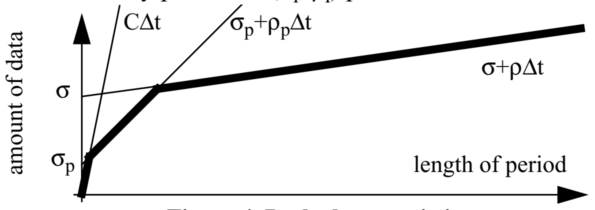

Figure 4. Peak characteristics

7. Case study

A video playback application has been studied to validate the usefulness of the presented theory. The video playback application is implemented on a shared memory architecture.

7.1 Case study description

The processors and hardware IP use the AXI interface protocol [8] to access DRAM. There are four processing elements (PEs), two programmable processors, an ARM CPU and a TriMedia DSP, and two hardware IPs, a scaler and a display controller (DC). A block diagram is shown in Figure 5. All communication between the PEs goes via shared memory. The ARM reads a bitstream from the flash memory (not shown in Figure 5), demultiplexes the bitstream into video and audio and handles the audio decoding. The video bitstream is written into DRAM. The TriMedia runs a video decoder that reads the bitstream and decodes it into raw video frames of size 720x576 @ 50 frames per second in 8 bit YUV4:2:0 video format (1.5 bytes/pixel). The scaler reads the raw video from DRAM, scales the video to the display size (800x600), converts the video format into 32 bit RGB, and writes the scaled and converted video data back into DRAM. The DC reads the video from the DRAM and sends it to the display. The frame rate is 50 frames per second in all the steps.

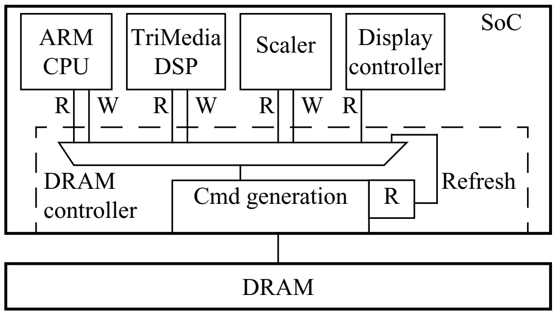

Figure 5. Block diagram of case study SoC+DRAM

There are 7 *request* flows in the case study. The ARM reads (1) and writes (2), the TriMedia reads (3) and writes (4), the scaler reads (5) and writes (6), and the DC reads (7). In the ARM and TriMedia read flows, both instructions and data reads are included. The characteristics are given in Table 1. The packet size is L=n_R B for all request flows.

In this case study, the ρ and σ of the flows from the ARM and the TriMedia are estimated based on clock frequency, cache miss rates, and basic characteristics of bitstream demultiplexing and video decoding. The ρ and σ of the flows from the scaler and the DC are calculated based on frame rate, resolution, and pixel formats. The flows from the scaler and the DC are strictly periodic, so that $\sigma = L\,(1\text{-}\rho/C)$. The transaction sizes (TS) are derived from cache line sizes from the programmable processors and sizes of the local memories in the hardware IPs.

Table 1. Flow characteristics and latency constraints

Flow	Type	σ [B]	ρ [kB/s]	TS [B]	LC	window	type
1_R	Read	$4n_R$	$189.9n_R$	32	6.00 ms	20 ms	round
2_R	Write	$2n_R$	$31.25n_R$	32	3.00 µs	1	one
3_R	Read	$4n_R$	$320n_R$	128	8.00 ms	20 ms	round
4_R	Write	$18.4n_R$	$243n_R$	128	3.00 µs	1	one
5_R	Read	$0.99n_R$	$243n_R$	128	4.11 µs	1	round
6_R	Write	$0.99n_R$	$750n_R$	128	3.00 µs	1	one
7_R	Read	$0.99n_R$	$750n_R$	128	2.66 µs	1	round

The degree of flows 1 and 3 is 1 because only one transaction is busy at any point in time. Flow 4_R has $\sigma_p = 0.99\ n_R$ B, $\rho_p = n_R$ MB/s. This is based on the assumption that the copy back unit of the data cache is regulated to write maximally one cache line per 1 µs.

The latency constraints (LC) on the read flows for the ARM and the TriMedia are derived from allowed stall cycles to be able to finish the processing on time. The LC on the read flows from the scaler and the display controller are derived from the sizes of local memories in the HW IPs. The LC on the write flows are derived from the bandwidth, the sizes of the write back buffer memories and the implementation of the synchronization scheme. The LC are given in Table 1. Window refers to a time window, a '1' indicates that the requirement holds per transaction. The type can be round-trip latency (round) or one-way latency (one).

The SoC AXI interconnect runs at 100 MHz and has 8 byte wide data connections. The DRAM controller runs also at 100 MHz and uses a 4 byte wide DDR interface (raw capacity is 800 MB/s). The request connections run at 100 MHz and are wide enough to transfer one request per clock cycle. All four PEs have private AXI channels to the memory controller. In AXI, read requests and write requests are independent, so each flow has its own channel to the DRAM controller. There is thus only one single point of arbitration in the system. Round robin (RR) is used as arbitration policy. It takes two clock cycles (20 ns) for a request to travel from a PE to the memory controller and it also takes two clock cycles for a response to travel from the memory controller back to the PE.

A 128 byte AXI transaction is split up into four DRAM bursts. The AXI transactions are issued on aligned addresses so that all DRAM bursts from one AXI transaction reside in the same DRAM row. This means that precharge and activate is not needed in the middle of one AXI transaction. From the DRAM specification [9] we can compute that a 128 B read transaction including activate and following precharge keeps the memory controller busy for 22 cycles. Similar calculations for 32 B read gives 10 cycles, for 32 B write 13 cycles, and for 128 B write 25 cycles.

7.2 Modelling and results

The DRAM is modelled according to Section 5.2. A 128 B read request is stretched to an AMP of size 22 x 8 B = 176 B. A 128 B write request is stretched to a 200 B AMP, a 32 B read request is stretched to a 80 B AMP, a 32 B write request is stretched to a 104 B AMP, and the refresh generator generates AMPs of 80 B because a refresh takes 10 cycles. In the case study, the request regulator is only applicable for flow 2 because no other flows allow requests close to each other. For flows 1 and 3 that is guaranteed by the degree and for flow 4 it is guaranteed by the peak characteristics.

The request regulator for flow 2 has the following D_{MR} and other parameters: $L_{out} = 104$ B, $\rho_{in} = 31250\ n_R$ B/s, $\sigma_{in} = 2\ n_R$ B, $\rho_q = 7692307\ n_R$ B/s, $\sigma_q = 0.9231\ n_R$, and $D_{MR} = 120$ ns according to Equation (5) and the other formulas presented in Section 5.1.

The characteristics of the AMP flows at the memory multiplexer, after request regulation and stretching, are computed from the sizes of the AMPs and the characteristics of the request flows according to the formulas in Section 5.1. The flows are described in Table 2. Flow 8 models refresh. Additional information is that flows 1_M and 3_M have degree 1 and flow 4_M has $\sigma_p = 150$ B, $\rho_p = 200$ MB/s.

Table 2. AMP flow characteristics

Flow	σ [B]	ρ [MB/s]	L [B]
1_M	318.6	15.192	80
2_M	207.6	3.250	104
3_M	692.2	56.320	176
4_M	3670.1	48.600	200
5_M	167.0	42.768	176
6_M	162.5	150.000	200
7_M	148.3	132.000	176
8_M	79.0	10.240	80

The memory multiplexer delay accounts for the largest share of the transaction delays. Because all links have the same capacity, and RR is an LFCFS policy, Equation (2) extended to 8 flows can be used. That gives the bounds in the second column in Table 3, which are not within the requirements. Tighter bounds are derived according to Section 6.1. The numbers from that analysis are shown in the third column in Table 3. Also those bounds are not within the requirements. Even tighter bounds can be given by including the peak characteristics of flow 4 and the degree of flows 1 and 3 as described in Section 6.2 and Section 6.4. Those bounds are shown in the fourth column in Table 3. E.g. for flow 2, the second packet can arrive at 130 ns after the start of the LBP. In the worst case it is serviced 2530 ns after the start of the LBP. The delay is thus 2530-130=2400 ns.

Table 3. Delays per request [ns]

Flow	Equation (2)	RR	Peak and degree	D_{total}
1	14872	4560	1390	1490
2	15531	2400	2400	2640
3	12884	3860	1270	1520
4	9678	5440	1490	1720
5	13970	1270	1270	1520
6	10902	1240	1240	1470
7	11337	1270	1270	1520
8	15378	1390	1390	1490

The total transaction delays consist of the wire delays and the memory delays. The memory execution delays (D_{ME}) are determined according to Section 5.2. Flow 1 uses critical word first, all the others do not. D_{MR} and D_{MM} have been presented above. The worst case transaction delays D_{total} are presented in column 5 in Table 3.

Flows 1 and 3 have constraints for the aggregate delay of the transactions within a 20 ms time window. As explained in Section 6.3 a straightforward bound on consecutive transactions is to multiply the number of transactions with the delay per transaction. E.g. for flow 3 there can be 6404 transaction with a delay of 1520 ns, which gives an aggregate delay of 9.73 ms. This does not meet the latency constraint for flow 3. By making use of Equation (7) the aggregate delay for flow 3 can be limited to 627*1520 ns + 1934*1390 ns + 1241*1290 ns + 1059*1190 ns + 18*970 ns + 1525*680 ns = 7.56 ms. This meets the latency constraint for flow 3.

8. Discussion and Conclusions

We have shown how network calculus can be applied to the performance analysis of SoC memory accesses. We focused on DRAM accesses because they are the most challenging. The approach we have taken is also applicable to on-chip and off-chip SRAMs and memory mapped registers.

The challenge is to derive tight bounds with concise models and simple analysis. There is a trade-off between tightness and simplicity. We have added information in the form of arbitration policy, degree, and peak characteristics to the basic (σ, ρ) traffic descriptions to improve the tightness. This tightens the bounds significantly at an acceptable modelling and analysis effort. For the presented case study all the presented techniques were needed to show that the design fulfils all requirements.

We conclude that it is possible to apply network calculus for worst-case analysis of memory access performance in SoCs. Using the presented modelling approach and tight analysis considerations, sufficiently tight bounds can be derived. Future work is to include this analysis approach in a system performance verification environment.

References

[1] R.L. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation and part II: Network Analysis. In *IEEE Transactions on Information Theory*, vol. 37, no.1, pp. 114-141, January 1991.

[2] J.-Y. LeBoudec. Network Calculus. Springer Verlag, no. 2050, LCNS, 2001.

[3] 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.

[4] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. *Proc. of the International Symposium on Circuits and Systems (ISCAS)*, pp. 101-104, vol. 4, 2000.

[5] M. Jersak et al. Performance Analysis for Complex Embedded Applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC*, vol. 45, issue 1/2, pp. 33-49, 2005.

[6] Y.-T. S. Li et al. Performance Estimation of Embedded Software with Instruction Cache Modeling. IEEE pp. 380-387,1995.

[7] J. Staschulat et al. Analysis of Memory Latencies in Multi-Processor Systems. Workshop on WCET, pp. 33-36, 2005.

[8] AMBA 3 Specification & Assertions. http://www.arm.com/products/solutions/axi_spec.html

[9] Micron, Mobile DDR SDRAM MT46H16M32LF. http://download.micron.com/pdf/datasheets/dram/mobile/MT46H32M16LF.pdf