An improved algorithm for slot selection in the Æthereal Network-on-Chip

Radu Stefan
Delft University of Technology
R.A.Stefan@tudelft.nl

Kees Goossens
Eindhoven University of Technology
K.G.W.Goossens@tue.nl

ABSTRACT

The rapid development in the electronics industry leads to a design process dominated by time-to-market constraints. The balance is shifted from logic design to packaging of already existing IP which results in a search for solutions for interconnecting the IP blocks. Networks-on-chip allow the rapid development a scalable interconnect and with the use of Circuit switching they can also provide guarantees for the speed of communication between IPs. In the current paper we demonstrate an improvement in the allocation algorithms for a Time-Division-Multiplexing Circuit-Switching scheme. We prove our algorithm to be optimal and we find that it provides an improvement of up to 26.7% compared to the previously proposed algorithm. The gain is more attractive as it comes at no cost for the actual hardware implementation.

1. INTRODUCTION AND BACKGROUND

The exponential growth in integration density that the electronics industry has sustained for the last five decades has led to a corresponding increase in design complexity. Although advances in design tools have mitigated to some extent the increased cost of designing more complex circuits the problem is by no means solved.

One way to address design complexity is through IP reuse. Current designs and to an even larger extent future designs [1] consist of IP blocks that have been previously deployed and verified in other products.

It not always certain though that an IP block which was tested and validated in isolation also works when connected together with other IPs. Arbitration to shared resources may cause for example delays which interfere with the proper functioning of the modules. Putting parts together and interconnecting them is therefore an important step in the design process and more so as the number of components increases.

Networks-on-Chip or NoCs are the emerging paradigm for the on-chip interconnect [5, 3]. They promote modularity, and IP reuse, they support scalability and in some cases they provide guarantees on communication performance which can be used to verify that an IP will perform properly once integrated in a larger system.

In this study we make use of the Æthereal Network-on-Chip which employs a Time-Division-Multiplexing (TDM) Circuit-Switching mechanism to provide bandwidth and latency guarantees. We show an improvement in the algorithm for TDM slot selection as we will describe in the following sections.

The core of the Æthereal network provides transport-level services [6] in the form of guaranteed bandwidth and latency communication channels between IPs, while Network Interface Shells at the boundary of the network are in charge of translating the bus protocol used by the IP to the internal network packet format (Figure 1).

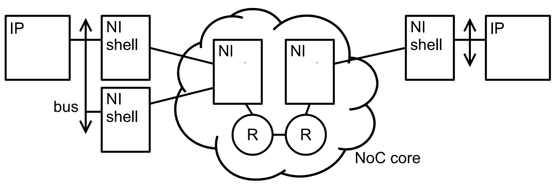

Figure 1: Æthereal Network

In the circuit switching implementation communication channels are established between connected IPs and receive exclusive use of some of the network resources throughout their entire lifetime.

The basic resource allocated to communication channels is the TDM time slot. The bandwidth of each link is split into a number of timeslots (the same number for each link) and internal network state tables describe who has access to each of the slots according to a schedule (a TDM slot wheel).

The sequence of allocated slots determines communication parameters like bandwidth and the scheduling latency as we will explain in section 3. In Æthereal the other component of latency, the network traversal latency is directly proportional to the path length as the latency at each hop is always fixed.

In this paper we present an improved algorithm for the allocation of slots. The paper is organized as follows. In the following section we present related work. The proposed TDM slot allocation algorithms are presented in Section 3. Experimental results are presented in Section 4 while the last section presents our conclusions.

2. PREVIOUS WORK

The problem or routing in NoCs has been widely studied [2, 4, 10, 11]. Our algorithm is related to the routing problem but instead of spatial allocation it deals with allocation along the time axis. It is expected to be used in conjunction with an algorithm for route selection in a TDM network for optimizing latency.

Our technique applies in general to NoCs using TDM, like Nostrum [15], aSoC [12] and [19]. Although some implementation details like the computation of header overhead are specific to the Æthereal implementation we believe that similar problems arising in other implementations could be solved by the same algorithm.

A related algorithm achieving both path and slot allocation is presented in [14]. However it only optimizes for bandwidth and does not consider latency nor the complex header overhead found in Æthereal .

The algorithm previously proposed for solving this problem is presented in [7], we use this to compare the performance of our algorithm. An algorithm improvement involving path rip-up and reallocation on top of the normal Æthereal algorithms is presented in [18]. While we do not directly investigate this technique we consider our algorithm to be compatible with it.

An online allocation method for the Æthereal network is proposed in [16] however the details of the allocation algorithm are not provided and could not be included in a comparison. Our algorithm is more complex than algorithm in [7], but it still runs in polynomial time and is not entirely unsuitable for online allocation.

An analysis of communication latency after the slots have been selected is provided in [17], while [9] details the relation between communication performance and the overall application behavior.

In a similar network implementation with more relaxed constraints on slot alignment, a graph coloring algorithm is proposed by [13] to solve a slot allocation problem. Our algorithm is to a large extent motivated by the restriction that in Æthereal slots need to be forwarded on the next link without delay. However other TDM networks may choose to do this even when it is not mandatory, for improving latency.

3. PROPOSED ALLOCATION ALGORITHM

The allocation procedure in the Æthereal tool-flow consists of two steps. The first one, detailed in [8] is in charge of finding proper paths. The second is typically executed inside the optimization loop of the first and consists of selecting timeslots on that path so that the latency and bandwidth constraints are met.

We do not attempt here to detail the pathfinding algorithm, but instead focus on the method of slot-selection.

The slot selection algorithm is given the set of available slots on a path has the task of identifying a minimal subset of slots that provide the required bandwidth and latency.

For simplifying the implementation, the bandwidth *bw* is always expressed in terms of words/slot table revolution. The number of words per slot table is derived from the bandwidth of a link, the size of the slot table and the required bandwidth, by rounding up to the nearest integer value.

Particularities of the Æthereal implementation slightly complicate the computation of delivered bandwidth. In particular, for consecutive slots belonging to the same channel, headers need to be inserted periodically to allow the transmission of credits for flow control. In our tests we assumed the period to be of 3 slots which is also the default value. The algorithm however is in no way tied to this particular value and can even ignore the presence of headers entirely.

The maximum latency l, expressed in units of the length in time of a slot is obtained by subtracting from the latency required by the application the latency that is due to network traversal, which is computed from the path length.

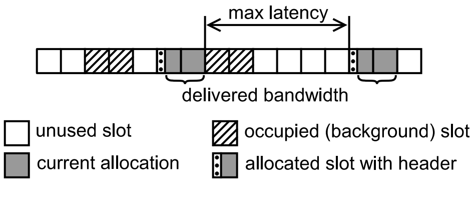

Figure 2: Problem formulation

The previously proposed algorithm addresses the two requirements (the latency requirement and the bandwidth requirement) separately. Furthermore it is a greedy algorithm taking optimal decisions only locally. We propose instead an algorithm which simultaneously optimizes according to both criteria and furthermore we show it to be optimal in that it uses a minimal number of slots.

A formal description of the algorithm is given in Algorithm 1.

Algorithm 1: Optimal slot selection

```
Data: set of available slots $A \subset S = \{s_1..s_n\}$
$l$ maximum allowed latency measured in slot periods
$\mathcal{B}_r$ required bandwidth
Result: $\mathcal{A}$ which minimizes $|\mathcal{A}|$ while satisfying bandwidth
        and latency constraints

if $\mathcal{B}_r > \mathcal{B}_A \vee \mathcal{L}atency(A) > l$ then
    fail;
end
$small \leftarrow slotSize - headerSize$;
$large \leftarrow slotSize - headerSize$;
solution $\mathcal{A} \leftarrow A$;
$\mathcal{B}_\mathcal{A} \leftarrow \mathcal{B}_A$;
for $\forall i \in \{1, 2, ...n\}$ do
    reorder original $A, S$ to start with slot $s_i$, $s_i$ becomes $s_1$;
    if $s_2 \in A$ then
        $\mathcal{A}_{2,1,1} \leftarrow \{s_2\}$;
        $\mathcal{B}_{\mathcal{A}_{2,1,1}} \leftarrow short$;
        for $\forall k \in \{3, 4, ...n\}$ do
            for $\forall i \in \{0, 1, 2\}, \forall j \in \{1..k-1\}$ do
                $\mathcal{A}_{k,i,j} \leftarrow \emptyset$;
                $\mathcal{B}_{\mathcal{A}_{k,i,j}} \leftarrow 0$;
            end
            i=1;
            for $\forall j \in \{2..k-1\}$ do
                /* adding one non-consecutive slot   */
                for
                $\forall x \in \{0, 1, 2\}, \forall y \in \{\max(2, k-l)..k-2\}$ do
                    if $\mathcal{B}_{\mathcal{A}_{k,i,j}} < \mathcal{B}_{\mathcal{A}_{y,x,j-1}} + small$ then
                        // update best known $\mathcal{A}_{k,i,j}$
                        $\mathcal{B}_{\mathcal{A}_{k,i,j}} \leftarrow \mathcal{B}_{\mathcal{A}_{y,x,j-1}} + small$;
                        $\mathcal{A}_{k,i,j} \leftarrow \mathcal{A}_{y,x,j-1} \cup s_k$;
                    end
                end
            end
            for $\forall i \in \{0, 1, 2\}, \forall j \in \{2..k-1\}$ do
                /* add consecutive slot   */
                $x = (i-1) \mod 3$;
                $gain \leftarrow large$;
                if $i = 1$ then
                    $gain \leftarrow small$;
                end
                if $\mathcal{B}_{\mathcal{A}_{k,i,j}} < \mathcal{B}_{\mathcal{A}_{k-1,x,j-1}} + gain$ then
                    // update best known $\mathcal{A}_{k,i,j}$
                    $\mathcal{B}_{\mathcal{A}_{k,i,j}} \leftarrow \mathcal{B}_{\mathcal{A}_{k-1,x,j-1}} + gain$;
                    $\mathcal{A}_{k,i,j} \leftarrow \mathcal{A}_{k-1,x,j-1} \cup s_k$;
                end
            end
        end
        for $\forall k \in \{n-l+1, 4, ...n\}$ do
            /* limiting the search to $n-l+1$ ensures
               that the latency limit is obeyed at
               wrap-around   */
            for $\forall i \in \{0, 1, 2\}, \forall j \in \{1..k-1\}$ do
                if $\mathcal{B}_r \leq \mathcal{B}_{\mathcal{A}_{k,i,j}}$ then
                    if $j < |\mathcal{A}| \vee (j = |\mathcal{A}| \wedge \mathcal{B}_\mathcal{A} < \mathcal{B}_{\mathcal{A}_{k,i,j}})$
                    then
                        /* note that $|\mathcal{A}_{k,i,j}| = j$   */
                        $\mathcal{A} \leftarrow \mathcal{A}_{k,i,j}$;
                        $\mathcal{B}_\mathcal{A} \leftarrow \mathcal{B}_{\mathcal{A}_{k,i,j}}$;
                    end
                end
            end
        end
    end
end
```

We first restrict solutions to a list of slots starting with one non-selected slot followed by one selected slot. By iterating over all rotations of the slot table, with wrap-around, we ensure the coverage of the entire solution space, one exception being a table with all slots selected, which is treated as a separate case.

We then build a set of partial optimal solutions. For each slot position k we build the set of partial optimal solutions with for each: (modulo 0..2 consecutive slots before slot k and $1..k-1$ slots used in total) Figure 3. The solutions are optimal in that they provide the best bandwidth given the constraints. All partial solutions obey the latency limit and can be constructed from the partial solutions for the nodes $n - latency + 1..n$.

For $k = 1$ no partial solution can be constructed since we already assumed the first slot is not used. For $k = 2$ only one partial solution exists which is $\{s_2\}$ and therefore this solution is optimal considering the given restrictions.

Let us formalize the previous algorithm description. We will already assume solutions starting with one unselected and one selected slot, the solution with all slots selected is trivial.

Let S be the set of all slots $\{s_1, s_2, ..s_n\}$, and let $A \subseteq S$ be the set of available slots $A = \{s_i \in S | s_i \text{ is not occupied}\}$. Let $A_{k,i,j}$ be a subset of $\{s_2, s_3..., s_k\}$ with $s_k \in A_{k,i,j}$, having exactly j elements ($|A_{k,i,j}| = j$ and ending with q selected slots, where $q \bmod 3 = i$, in other words $s_{k-q} \notin A_{k,i,j}$ and $\{s_{k-q+1}, s_{k-q+2}..., s_k\} \subset A_{k,i,j}$. 3 is the maximum number of slots after which the header needs to be repeated. It can be replaced in the algorithm by any other constant.

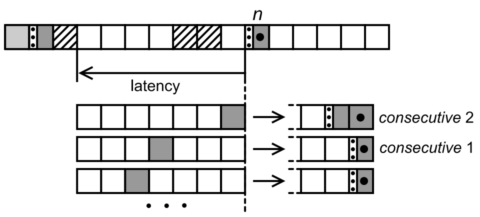

Figure 3: Building partial solutions

The reason behind the classification of partial solutions by their modulo 3 number of ending slots is that it enables us to compute the bandwidth obtained by attaching one additional slot at the end of the partial solution, that is, if the bandwidth delivered by $A_{k-1,i,j-1}$ is $\mathcal{B}_{A_{k-1,i,j-1}}$, the bandwidth delivered by $A_{k-1,i,j-1} \cup s_k$ is:

$$\mathcal{B}_{A_{k,i,j}} = \begin{cases} \mathcal{B}_{A_{k-1,i,j-1} \cup s_k} + \text{slotSize-headerSize} & \text{when } i = 0 \\ \mathcal{B}_{A_{k-1,i,j-1} \cup s_k} + \text{slotSize} & \text{otherwise} \end{cases} \tag{1}$$

When attaching one slot to a solution in which the last slot is not selected (a non-consecutive slot, line 29 in Algorithm 1), we always pay the header penalty and the number of slots at the end of solution becomes 1.

$$\mathcal{B}_{A_{k,1,j}} = \mathcal{B}_{A_{k-m,i,j-1} \cup s_k} + \text{slotSize-headerSize} \quad \text{where } m > 1 \tag{2}$$

It is easy to see that any solution having at least two slots $A_{k,i,j}; j \geq 2$ can be build from a solution $A_{x,y,j-1}$ by adding a new slot on the position k, and the delivered bandwidth can be computed using one of the equations 1, 2.

3.1 Proof of optimality

Let us denote $\mathcal{A}_{k,i,j}$ a set $A_{k,i,j}$ that is optimal in that for the given k, i, j it provides the largest bandwidth. We argue that $\mathcal{A}_{k,i,j}$, if it exists can only be obtained by adding a slot k to one optimal set $\mathcal{A}_{x,y,j-1}$. Indeed, if that was not the case, then $\mathcal{A}_{k,i,j}$ would be obtained from a non-optimal $A_{x,y,j-1}$ as $A_{x,y,j-1} \cup \{s_k\}$ and $\mathcal{B}_{\mathcal{A}_{k,i,j}} = \mathcal{B}_{A_{x,y,j-1}} + q$ where q is a constant dependent only on x and k, derived from equations 1, 2. Note that $k - x < latency$ to obey the latency requirement.

But since $A_{x,y,j-1}$ is not optimal $\exists \mathcal{A}_{x,y,j-1}$ so that $\mathcal{B}_{\mathcal{A}_{x,y,j-1}} > \mathcal{B}_{A_{x,y,j-1}} \Rightarrow \mathcal{B}_{\mathcal{A}_{x,y,j-1} \cup \{s_k\}} + q > \mathcal{B}_{\mathcal{A}_{k,i,j}}$ and $\mathcal{A}_{k,i,j}$ is not optimal, which would contradict the hypothesis.

It results from here that if we generate all feasible $\mathcal{A}_{x,y,j-1}$ sets (or at least one set for each $(x, y, j-1)$) we can generate, if it exists, any $\mathcal{A}_{k,i,j}$ set.

An optimum to our original problem, that is, a subset of S which satisfies the latency bound l, and has a minimum required bandwidth bw, can always be expressed as a set $A_{k,i,j}$, by properly selecting values for k, i and j, but since $\mathcal{A}_{k,i,j}$ uses the same number of slots, j and it provides bandwidth as high as any of the $A_{k,i,j}$ sets, $\mathcal{A}_{k,i,j}$ is also an optimal solution, with the added benefit that among the solutions that use j slots it also provides the highest possible bandwidth, which was not one of the original problem requirements.

In our algorithm, the optimum is found by enumerating all $\mathcal{A}_{k,i,j}$ sets and selecting the one which:

1. Provides the necessary bandwidth
2. Obeys wrap-around latency requirement
3. Has the lowest j value (number of used slots)
4. For the lowest j value has the highest $\mathcal{B}_{\mathcal{A}_{k,i,j}}$

We have also verified the optimality of our solution experimentally by comparison against an exhaustive search for slot table sizes up to 24.

3.2 Algorithm complexity

Consider the following notation for the calculation of complexity: n is the number of slots in the slot table; m is the number of free slots $m \leq n$, worst case $m = n$; l is the maximum distance between slots; p is the maximum number of consecutive slots after which the header needs to be repeated.

The dynamic programming algorithm is run m times for each rotation of the slot table in which the second slot is available (as explained previously). Each of these runs involves building a table of partial solutions $(A_{k,i,j})$ of size $n^2 * p$. When slot k is available (which happens for m of the slots) $A_{k,1,j}$ is computed based on $l * p$ other values, and $A_{k,i,j}$ with $i \neq 0$ is computed based on one other value. The complexity for computing the table is $O(m^2 * l * p)$ and the total algorithm complexity $O(m^3 * l * p)$ which is in the worst case $O(n^3 * l * p)$. The memory complexity is dominated by the size of the tables mentioned and is $O(n^2 * p)$.

This is higher than the original algorithm which had a complexity of $O(n)$, in practice however it does not represent a problem since allocation is typically performed at design time.

4. EXPERIMENTAL RESULTS

In this section we compare our proposed algorithm with the original greedy algorithm. We perform the comparison using sets of occupied and unoccupied slots, and iterating over all feasible bandwidth/latency pairs for each pattern. At each design point we employ 1000 random samples of background traffic, and we report the improvement of our algorithm against the algorithm in [7].

The average improvement for a slot table sizes of 32 is plotted in figure 4 against the background utilization (the number of slots that were already marked as occupied when the slot selection algorithm was run). We have also performed experiments for table sizes of 8-40 but due to lack of space we cannot present them here.

The improvement can take two forms: in some cases, the dynamic programming algorithm can deliver the required bandwidth and latency while using fewer slots than the greedy

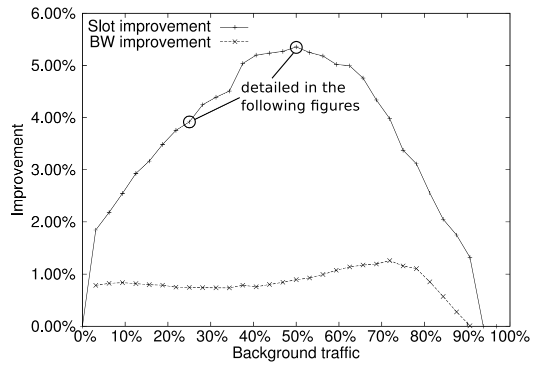

Figure 4: Improvement in slot utilization vs background utilization

algorithm. We call this slot improvement. When not producing slot improvement, there is still a chance that the dynamic programming algorithm delivers more bandwidth than is required due to the granularity of slots and this bandwidth is in some cases higher than the one provided by the greedy algorithm. This is called Bandwidth improvement in the figures.

Obviously little gain can be obtained when the slot table is essentially empty or when it is completely full. Some improvement exists though for a completely empty table because the original algorithm does not properly take into account bandwidth gain at wrap-around. The largest gains are in the middle section of the interval, which we consider likely to be found in practice.

This is however the gain averaged over all requested latency/bandwidth values. Figures 5 and 6 detail the 25% and 50% background utilization scenarios over the range of requested latency and bandwidth values. We can see that for very tight latency constraints there is hardly any gain since there is very little flexibility in choosing the needed slots. Also very little gain is made when the required bandwidth is very low or very high. For the first case it means that for latency-only constraints the initial greedy algorithm is performing very well, for the latter obviously when the entire bandwidth needs to be allocated only one solution exists and that consists of allocating all the available slots.

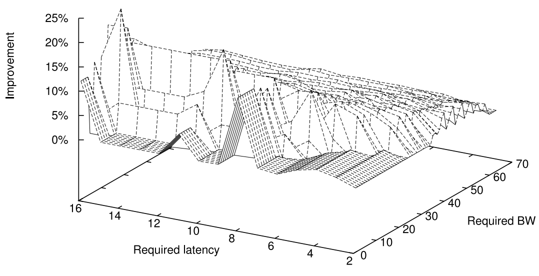

Figure 5: Detailing improvement under 25% utilization vs. latency/bw requirements

It is noticeable that for the lower bandwidths the graph curve is independent of bandwidth. This is because the slots allocated because the latency constraint are sufficient to also satisfy the bandwidth constraint.

5. CONCLUSIONS

We conclude with the observation that the original algorithm was on average quite efficient, but the proposed algorithm can nevertheless show some improvement and in addition by being optimal it provides a bound on performance. The new algorithm does not incur any cost on the hardware implementation, the disadvantage, if any, is a slight increase in algorithm running time which is completely negligible if performed at design time.

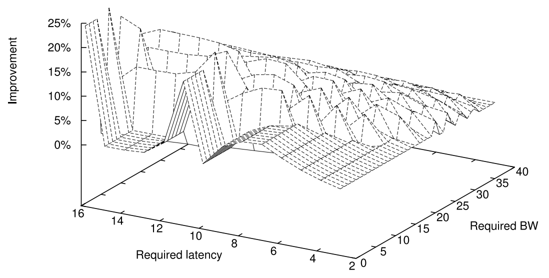

Figure 6: Detailing improvement under 50% utilization vs. latency/bw requirements

6. REFERENCES

[1] The international technology roadmap for semiconductors, 2009. www.itrs.net.

[2] G. Ascia *et al.* A new selection policy for adaptive routing in network on chip. In *EHAC*, 2006.

[3] L. Benini and G. De Micheli. Networks on chips: a new SoC paradigm. *Computer*, 35(1), Jan 2002.

[4] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE TPDS*, 4(4), 1993.

[5] W. J. Dally and B. Towles. Route packets, not wires: On-chip inteconnection networks. In *DAC*, 2001.

[6] J. D. Day and H. Zimmermann. The OSI reference model. 1995.

[7] A. Hansson. *A Composable and Predictable On-Chip Interconnect.* PhD thesis, TU Eindhoven, 2009.

[8] A. Hansson *et al.* A unified approach to mapping and routing on a NoC for both best-effort and guaranteed service traffic. *VLSI Design*, 2007.

[9] A. Hansson *et al.* Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis. *IET-CDT*, 3(5), 2009.

[10] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *DATE*, 2003.

[11] J. Hu and R. Marculescu. DyAD: Smart routing for networks-on-chip. In *DAC*, 2004.

[12] J. Liang *et al.* aSOC: A scalable, single-chip communications architecture. In *PACT*, 2000.

[13] J. Liu *et al.* Interconnect intellectual property for network-on-chip (NoC). *J. Syst. Arch.*, 2004.

[14] Z. Lu *et al.* TDM Virtual-Circuit configuration for Network-on-Chip. *TVLSI*, 2008.

[15] M. Millberg *et al.* Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *DATE*, 2004.

[16] O. Moreira *et al.* Online resource management in a multiprocessor with a network-on-chip. In *ACM-SAC*, Seoul, Korea, 2007.

[17] A. T. Nelson. Conservative application-level performance analysis through simulation of a multiprocessor system on chip. Master's thesis, TU Eindhoven, 2009.

[18] S. Stuijk *et al.* Resource-efficient routing and scheduling of time-constrained streaming communication on NoC. *J. of Sys. Arch.*, 2008.

[19] Y. Wang *et al.* Dynamic TDM virtual circuit implementation for NoC. In *APCCAS*, 2008.