Message scheduling to reduce AFDX jitter in a mixed NoC/AFDX architecture

Jérôme Ermont
Université de Toulouse/IRIT/Toulouse INP-ENSEEIHT
Toulouse
jerome.ermont@enseeiht.fr

Sandrine Mouysset
Université de Toulouse/IRIT/UPS
Toulouse
sandrine.mouysset@irit.fr

Jean-Luc Scharbarg
Université de Toulouse/IRIT/Toulouse INP-ENSEEIHT
Toulouse
jean-luc.scharbarg@enseeiht.fr

Christian Fraboul
Université de Toulouse/IRIT/Toulouse INP-ENSEEIHT
Toulouse
christian.fraboul@enseeiht.fr

ABSTRACT

Current avionics architecture are based on an avionics full duplex switched Ethernet network (AFDX) that interconnects end systems. Avionics functions exchange data through Virtual Links (VLs), which are static flows with bounded bandwidth. The jitter for each VL at AFDX entrance has to be less than 500 μs. This constraint is met, thanks to end system scheduling. The interconnection of many-cores by an AFDX backbone is envisioned for future avionics architecture. The principle is to distribute avionics functions on these many-cores. Many-cores are based on simple cores interconnected by a Network-on-Chip (NoC). The allocation of functions on the available cores as well as the transmission of flows on the NoC has to be performed in such a way that the jitter for each VL at AFDX entrance is still less than 500 μs. A first solution has been proposed, where each function manages the transmission of its VLs. The idea of this solution is to distribute functions on each many-core in order to minimize contentions for VLs which concern functions allocated on different many-cores. In this paper, we consider that VL transmissions are managed by a single task in each many-core. We propose to construct a scheduling table executed by this task using an Integer Linear Program. The access to the Ethernet interface is then only allowed to one VL leading to a significant reduction of the jitter.

Jérôme Ermont, Sandrine Mouysset, Jean-Luc Scharbarg, and Christian Fraboul. 2018. Message scheduling to reduce AFDX jitter in a mixed NoC/AFDX architecture. In *26th International Conference on Real-Time Networks and Systems (RTNS '18), October 10–12, 2018, Chasseneuil-du-Poitou, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3273905.3273929

1 INTRODUCTION

Aircrafts are equipped with numerous electronic equipment. Some of them, like flight control and guidance systems, provide flight critical functions, while others may provide assistance services that are not critical to maintain airworthiness. Current avionics architecture is based on the integration of numerous functions with different criticality levels into single computing systems (mono-core processors) [10]. Such an architecture is depicted in Figure 1. Computing systems are interconnected by an AFDX (Avionics Full Duplex Switched Ethernet) [1]. The End System (ES) provides an interface between a processing unit and the network.

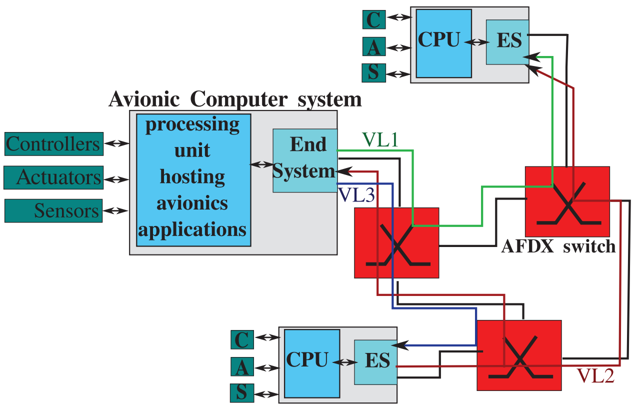


Figure 1: An AFDX network.

The continuous need for increased computational power has fueled the on-going move to multi-core architectures in hard real-time systems. However, multi-core architectures are based on complex hardware mechanisms, such as advanced branch predictors whose temporal behavior is difficult to master. Many-core architectures are based on simpler cores interconnected by a Network-on-Chip (NoC). These cores are more predictable [17]. Thus, many-cores are promising candidates for avionics architecture.

A typical many-cores architecture provides Ethernet interfaces and memory controllers. For instance, Tilera Tile64 has 3 Ethernet interfaces and 4 memory controllers [18], while Kalray MPPA has 8 Ethernet interfaces and 2 memory controllers [8].

An envisioned avionics architecture is depicted in Figure 2. A set of many-cores are interconnected by an AFDX backbone, leading to a mixed NoC/AFDX architecture. Avionics functions are distributed on these many-cores. Communications between two

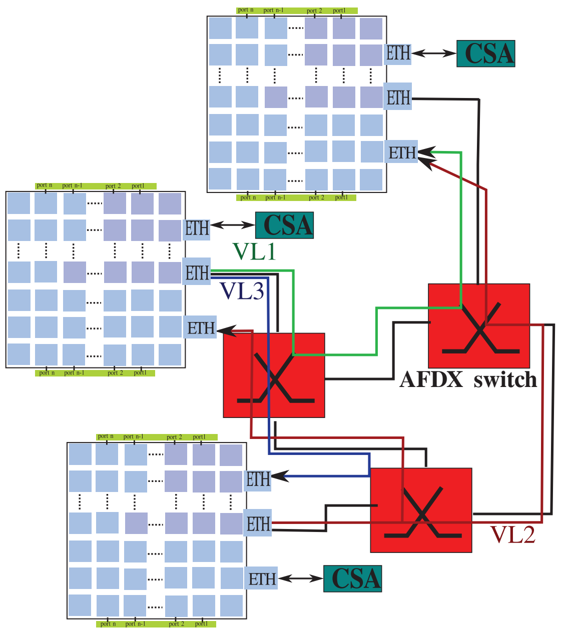


Figure 2: A mixed NoC/AFDX architecture

functions allocated on the same many-cores use the NoC, while the communications between two functions allocated on different many-cores use both the NoC and the AFDX. Main constraints on this communication are the following:

(1) end-to-end transmission delay has to be upper-bounded by an application defined value,
(2) frame jitter at the ingress of the AFDX network has to be smaller than a given value (typically 500 μs).

In single core architectures the latter constraint is enforced by the scheduling implemented in the End System. In many-cores architectures, frame jitter mainly depends on the delay variation between the source core and the source Ethernet interface. This variation is due to two factors. First, the frame can be delayed on the NoC by other frames transmitted between avionic functions. Second, the Ethernet controller can be busy, transmitting another frame. [2] proposes a mapping strategy which minimizes the first factor, i.e. the variation of this NoC delay. Each core is allocated at most one function. Each VL is managed by its source function.

In this paper, we mainly address the second factor. We propose a static scheduling of Ethernet transmissions, based on a table. Each transmission is allocated a periodic slot. The scheduling is managed by a dedicated core. The periodic slots are allocated to the applications in order to guarantee that the access to the Ethernet interface is allowed to one application. We formulate the mapping of the applications to the slots as an Integer Linear Program (ILP). The objectives of the approach are:

(1) to allocate the slots for the transmission of the outgoing flows within the respect of their periodic constraint ;
(2) to oversample the slots for outgoing flows with period $> n$ ms, with $n = 1, 2, 4, \ldots$

The remainder of the paper is as follows. Section 2 introduces current AFDX and NoC architectures. Section 3 explains the addressed problem. The new approach is described in Section 4. Section 5 shows the proposed scheduling solution on a case study and give some results compared to the distributed solution of [2]. Finally, Section 6 concludes with some future works.

2 SYSTEM MODEL

We summarize the main features of both a AFDX flows and many-cores considered in this paper.

2.1 AFDX flows

A VL defines a unidirectional connection between one source function and one or more destination functions. Each VL is characterized by two parameters:

- **Bandwidth Allocation Gap (BAG).** Minimal time interval separating two successive frames of the same VL. The value of the BAG is ranging from 1 to 128 ms.
- **L_{min} and L_{max}.** Smallest and largest Ethernet frame, in bytes, transmitted on the VL.

In current architectures, each ES performs a traffic shaping for each VL to control that frames are transmitted in accordance with BAG and authorized frame size. The queued frames, which are ready to be transmitted, are then selected depending on a strategy configured in the VL scheduler. Therefore, it is possible that more than one VL has a packet ready and eligible for transmission. In this case, a queuing delay (jitter) is introduced. This jitter, computed at the transmitting ES, is the time between the beginning of BAG interval and the first bit of the frame to be transmitted in that BAG. This jitter must not be greater than 500μs.

2.2 NoC Architecture and Assumptions

In this paper, we consider a Tilera-like NoC architecture, *i.e.* a 2D-Mesh NoC with bidirectional links interconnecting a number of routers. Each router has five input and output ports. Each input port consists of a single queuing buffer. The routers at the edge of the NoC are interconnected to the DDR memory located north and south of the NoC via dedicated ports. The first and last columns of the NoC are not connected directly to the DDR. Besides, the routers at the east side connects the cores to the Ethernet interfaces via specific ports. Many applications can be allocated on a NoC. Each application is composed of a number of tasks, where one core executes only one task. These tasks do not only communicate with each other (core-to-core flows), but also with the I/O interfaces, i.e. the DDR memory and Ethernet interfaces (core-to-I/O flows). These flows are transmitted through the NoC following wormhole routing [16], an XY policy and a Round-Robin arbitration. Besides, a credit-based mechanism is applied to control the flows. A flow consists of a number of packets, corresponding to the maximal authorized flow size on the NoC. Indeed, a packet is divided into a set of flits (flow control digits) of fixed size (typically 32-bits). The maximal size of a NoC packet is of 19 flits as in Tilera NoC. The wormhole routing makes the flits follow the first flit of the packet in a pipeline way creating a worm where flits are distributed on many routers. The credit-based mechanism blocks the flits before a buffer overflow occurs. The consequence of such a transmission model is that when two flows share the same path, if one of them is blocked, the other one can also be blocked. Thus, the delay of a flow can increase due to contentions on the NoC. The Worst-case Traversal Time (WCTT) of a flow can be computed using different methods proposed in the literature [5, 13]. In this paper, we choose RC_{NoC} [3] to compute the WCTT as it leads to tightest bounds of delays compared to the existing methods on a Tilera-like NoC. This method considers the pipeline transmission, and thus computes the maximal blocking delay a flow can suffer due the contentions with blocking flows.

3 PROBLEM STATEMENT

Actually, an outgoing I/O flow is transmitted following three steps:

(1) A core sends data to the nearest port of DDR memory,
(2) then it sends a DMA command to the Ethernet interface on a separate network. This DMA command indicates that all the data are in the DDR memory. This command is stored into a DMA command FIFO queue.
(3) When the Ethernet interface executes the DMA command, data packets are sent from the same port of DDR memory to the same Ethernet interface. The packets of an outgoing I/O flow will incur a contention with different types of communications on the NoC which could lead to a jitter.

Let us illustrate the delays of these steps with the example in Figure 3. Two VLs VL_1 and VL_2 are respectively generated by tasks $t1_{DDR}$ and $t2_{DDR}$. At the beginning of VL_1 first BAG period, $t1_{DDR}$ sends VL_1 data to the nearest port of DDR memory. This transmission can take up to $WCTT_{toDDR}$. Step 2 (transmission of the DMA command to the Ethernet interface) is done after this worst-case delay. Thus step 1 duration is constant and does not generate any jitter. Similarly, step 2 duration is assumed to be constant, since the DMA command is sent on a separate network. Thus all the jitter comes from step 3 (transmission of the data from DDR memory to the Ethernet interface). Considering VL_1 first BAG period in Figure 3, the jitter is the delay $d1$ of this transmission, which is between 0 and its worst-case duration. The jitter can be much higher. Indeed, for VL_1 second BAG period, the Ethernet interface is busy with VL_2 when it receives VL_1 DMA command. The delay due to VL_2 has to be added to the jitter, leading to an overall value of $d + d_{VL2} + d2$.

The goal is to avoid that the Ethernet interface is busy when it receives a DMA command (like for VL_1 second period in Figure 3).

4 A DEDICATED CORE FOR OUTGOING I/O FLOWS

We propose to dedicate a specific core of the NoC to the transmission of VL through the Ethernet interface. The idea is to execute only one task t_{DDR} for all the applications. This task can be executed on a core of the NoC that do not execute any other task. The transmission of the command of the DDR uses another internal network and does not impact the communications of other tasks. The behaviour of the task t_{DDR} is as follow:

(1) Reception of a message from the final task of the function: the data that need to be sent are in the DDR. In such a way, we do not change the behaviour of the functions presented in section 2.2. The transmission of the message is constant and can be considered as a part of the execution time of the sending task of the function.
(2) Transmission of a DDR command to the DDR. The corresponding data are then transmitted from the DDR to the Ethernet interface.

In this paper, we consider that the schedule of the DDR commands, and so the VLs sent by the functions, can be done by using a scheduling table.

4.1 A scheduling table

The goal of this method is to reduce the jitter induced by the transmission of other VLs from the memory to the Ethernet interface through the NoC and the transmission of these VLs through the Ethernet interface. The considered scheduling table is composed of slots of $31.25\mu s$. The global duration of the table is 128 ms. So the number of slots is 4096. The table is composed of 128 lines of 1 ms, each line contains 32 slots. A set of slots is allocated to each VL sent by the applications by considering the BAG duration: a VL will obtain a slot at exactly each BAG. Such a scheduling is represented in Table 1. In this example, we denoted by the name of the application the slot when the corresponding VL should be sent. As an example, a VL from the application App_1 has a BAG duration equal to 2 ms. It can be located in column 1 of lines 0, 2, . . . , 126. In the same manner, VL from the application App_4 has a BAG of 8 ms and is located in column 10 of lines 1, 9, 17, . . . , 121.

For each application App_i which send a VL_i, the number of allocated slots ω_i is defined as:

$$\omega_i = \left\lceil \frac{\mathrm{WCTT}(\mathrm{VL}_i) + \mathrm{ft}(\mathrm{VL}_i)}{\mathrm{sd}} \right\rceil \tag{1}$$

where $\mathrm{WCTT}(\mathrm{VL_i})$ is the WCTT of VL_i from the DDR to the Ethernet interface, $\mathrm{ft}(\mathrm{VL_i})$ is the transmission delay of the frame through the Ethernet interface and $\mathrm{sd} = 31.25\mu s$ is the slot duration. The $\mathrm{WCTT}(\mathrm{VL_i})$ depends on the contention on the NoC, while $\mathrm{ft}(\mathrm{VL_i})$ depends on the size of the frame that is transmitted by the Ethernet interface. For example, if we consider that the WCTT of VL_2, sent by App_2, from the memory to the Ethernet interface is 30 μs and the transmission of the VL_2 by the interface needs 120 μs, the resulting value of ω_2 is 5 as depicted in Table 1.

This scheduling guarantees that the VLs are sent from the memory to the Ethernet interface at different times, leading to a reduction of the jitter induced by the transmission of other VLs. In such a way, the only delay that a VL can suffer when it is transmitted from the DDR is due to the interferences from the transmission of internal communications through the NoC that share the same path as the VL.

The goal is now to define this scheduling table, *i.e.* the allocation of the slots to the applications.

4.2 Allocation of the slots to applications

We propose an allocation solution with two objectives:

(1) The main objective of the allocation of the slots is to guarantee that the VLs sent by the applications respect their BAG constraints. The approach needs to guarantee that an application can send a data at exactly every BAG period and will find the next transmission slot at exactly one BAG from the

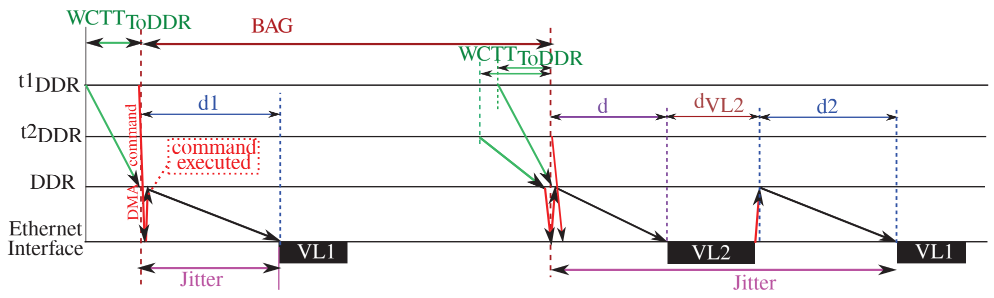


Figure 3: A possible transmission on a given VL.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App3	🟦	🟦	🟦	🟦	🟦	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧						
1	App2	🟪	🟪	🟪	🟪						App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
2	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨																				
3	App2	🟪	🟪	🟪	🟪																											
4	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App3	🟦	🟦	🟦	🟦	🟦														
5	App2	🟪	🟪	🟪	🟪																											
6	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨																				
7	App2	🟪	🟪	🟪	🟪																											
8	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App3	🟦	🟦	🟦	🟦	🟦	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧						
9	App2	🟪	🟪	🟪	🟪						App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩												
10	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨																				
11	App2	🟪	🟪	🟪	🟪																											
12	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App3	🟦	🟦	🟦	🟦	🟦														
13	App2	🟪	🟪	🟪	🟪																											
14	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨																				
15	App2	🟪	🟪	🟪	🟪																App6	🟥	🟥	🟥	🟥	🟥	🟥					
⋯																																
127	App2	🟪	🟪	🟪	🟪																App6	🟥	🟥	🟥	🟥	🟥	🟥					

Table 1: Scheduling table of the VL$_i$ sent by each application App$_i$ (VLs are represented by their sender applications).

current transmission. For example, if App$_1$ starts its transmission at slot located in column 5 of line 2, then App$_1$ can send a new data 2ms (the BAG of VL$_1$ is 2ms) later, at slot located in column 5 of line 4, as depicted in Table 1.

(2) In Table 1, App$_6$ can send VL$_6$ at column 20 in line 15. Considering the scheduling and the execution of App$_6$ in the many-core, VL$_6$ can be ready to be sent at 100μs, for example. So, the transmission of this VL through the Ethernet interface will start 15.5 ms later (15 lines of the table is 15 ms, 16 slots of 31.2μs is 0.5 ms). Thus, we propose to reduce this delay by allocating more transmission slots for each VLs. Our second objective is then to allocate slots every n ms for VLs, with $n = 1, 2, 4, \ldots$ In our approach, we propose to fix n to the minimum value of the BAG of the VLs sent by the applications. This is what we call oversampling.

We have then formulated these objectives into an ILP. The proposed solution is a sequence per line of slots allocated to the transmission of VLs that is repeated every n ms, with $n = 1, 2, 4, \ldots$ The mathematical definition is given in the following.

4.2.1 Formulation. With the notations in the table 2, we consider m different possible applications and their associated BAG, the scheduling table provides free slots that can be addressed for applications.

We have to treat the special case of BAG equal to 1ms because it impacts all the lines of the scheduling table and so the capacity of each line is reduced by the number of slots dedicated to the

Constants	
m	Number of applications
BAG_j	period of the application j noted App_j
ω_j	number of slots for application j
C	Capacity of each line of scheduling table by removing 1ms BAG
N	Number maximum of lines considered for allocation
Boolean Variables	
y_i	equal 1 if line i of the table is used in the solution, 0 otherwise
x_{ij}	equal 1 if application j is packed into line i of the table, 0 otherwise
Objective Function	
$\sum_{i=1}^{N} y_i$	minimize the number of lines to pack all the applications

Table 2: Notations for Bin Packing Problem

applications. So the capacity, noted C, in terms of free slots for each line of the scheduling table is defined as follows:

$$C = 32 - \sum_{j \in J} \omega_j, \tag{2}$$

where $J = \{j \in \{1, .., m\} | BAG_j = 1\}$. The capacity C should be in [|0,32|]. So negative values will indicate that the configuration cannot ensure the transmission.

From the computation of the real capacity of each BAG by taking into account the 1ms period of application, the allocation of each application can be formulated as an Integer Linear Program (ILP) called Bin Packing Problem [15]. The bin packing problem can be described informally as follows: by considering m applications, each having an integer number of slots ω_j (j=1,..,m) and a limited number of lines N of integer capacity C. The objective is to pack all the applications into the minimum number of lines in the table so that the total number of slots per application packed in any line of the table does not exceed its capacity. This will provide an optimal way to assign slots in the scheduling table by the applications and thus the filling of the table will permit transmitting applications with long BAG several times within its own BAG.

We have to fix the limited number of lines N to allocate all the applications. In fact, in order to repeat the most the allocations in the table, we take into account all the BAG_j (j=1..m), once. So the number N is equal to the minimum BAG among all the applications except 1ms (which are already considered by equation (1)) and is defined as follows:

$$N = \min_{j=1\dots m,\ BAG_j \neq 1} BAG_j. \tag{3}$$

Formally, let's introduce two binary decision variables y_i for $i \in \{1, .., N\}$ which indicates the considered line of the table and x_{ij}, for $i \in \{1, .., N\}$ and $j \in \{1, .., m\}$ which indicates the place of the application j in the line i of the table as defined in the table 2. From these notations, as the objective is to pack all the applications into the minimum number of lines in the scheduling table, the objective function will be formulated as:

$$\min \sum_{i=1}^{N} y_i$$

defined in Table 2. This function has to be minimized so that the total number of slots per application, noted $\sum_{j=1}^{m} \omega_j x_{ij}$ for a line $i \in \{1, .., N\}$, packed in any line $i \in \{1, ..N\}$ of the table does not exceed its capacity C and this constraint can be expressed for all the lines $i \in \{1, .., N\}$ as:

$$\sum_{j=1}^{m} \omega_j x_{ij} \leq C\, y_i, \forall i = 1, .., N.$$

Finally, as the aim is to fill the table by considering all the m applications, this leads to define these following constraints:

$$\sum_{i=1}^{N} x_{ij} = 1,\ \forall j = 1, .., m.$$

Thus, we can formulate the problem of filling the scheduling table by applications as the following ILP problem [14]:

$$\min \quad \sum_{i=1}^{N} y_i \tag{4}$$

$$s.t \quad \sum_{j=1}^{m} \omega_j x_{ij} \quad \leq C\, y_i, \forall i = 1, .., N \tag{5}$$

$$\sum_{i=1}^{N} x_{ij} \quad = 1, \forall j = 1, .., m \tag{6}$$

$$y_i \in \{0, 1\} \quad , \forall i = 1, .., N \tag{7}$$

$$x_{ij} \in \{0, 1\} \quad , \forall i = 1, .., N, \forall j = 1, .., m. \tag{8}$$

Objective function (3) means that the applications should be stored in a minimum of lines of table N defined by equation (2). Constraints (4) impose that the capacity of each line defined by equation (1) is not exceeded while constraints (5) ensure that all the applications are packed. This ILP problem is solved by using CPLEX [7].

4.2.2 *Example.*

Let's consider the application configuration of Table 1. It is composed of 6 applications. App$_2$ is the first mapped into the table as the BAG is equal to 1ms. With equation (1), the capacity C is equal to 27 free slots. Equation (2) computes the number of lines in which the applications has to be allocated is 2. The resulting scheduling table is given in Table 3. As all the applications can be packed in two lines, lines 0 and 1 are repeated in the table. So, applications App$_3$, App$_4$, App$_5$ and App$_6$, for which

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
1	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
2	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
3	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
4	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
5	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
6	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
7	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
8	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
9	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
10	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
11	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
12	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
13	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
14	App2	🟪	🟪	🟪	🟪	App1	🟨	🟨	🟨	🟨	🟨	🟨	App5	🟧	🟧	🟧	🟧	🟧	🟧	🟧												
15	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				
⋯																																
127	App2	🟪	🟪	🟪	🟪	App4	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	App3	🟦	🟦	🟦	🟦	🟦	App6	🟥	🟥	🟥	🟥	🟥	🟥				

Table 3: Scheduling table obtained by the mapping method.

the BAG values are 4, 8, 8 and 16 respectively, have a communication slot every 2ms. The waiting delay of these applications if they miss the first communication slot is then 2ms.

5 CASE STUDY

In this section, we evaluate the approach to realistic case studies.

5.1 Considered applications

The considered case study is composed of critical and non critical applications:

- **Full Authority Digital Engine (FADEC) application:** It controls the performance of the aircraft engine. It receives 30 KBytes of data from the engine sensors via an Ethernet interface and sends back 1500 Bytes of data to the engine actuators. The application FADEC$_n$ is composed of n tasks denoted t_{f0} to t_{fn-1}. t_{fn-1} is dedicated to send the commands to the engine actuators via the Ethernet interface. Except t_{fn-1}, all other tasks exchange 5 KBytes of data through the NoC. They also send 5 KBytes of data to t_{fn-1}. Figure 4a shows the tasks graph of FADEC$_7$. This graph illustrates the core-to-core and core-to-I/O communications between the tasks of the FADEC application.
- **Health Monitoring (HM) application:** It is used to recognize incipient failure conditions of engines. It receives through an Ethernet interface, a set of frames of size 130 KBytes and sends back 1500 bytes of data actuators. The application HM$_n$ is composed of n tasks, denoted t_{h0} to t_{hn-1}. The last task t_{hn-1} is dedicated to send the data actuators to the Ethernet interface. The task t_{hi} sends 2240 bytes of data to t_{hi+1} through the NoC, with $i \in [0, n-2]$. All these tasks finish their processing by storing their frames into the memory. Figure 4b shows the tasks graph of HM$_6$.

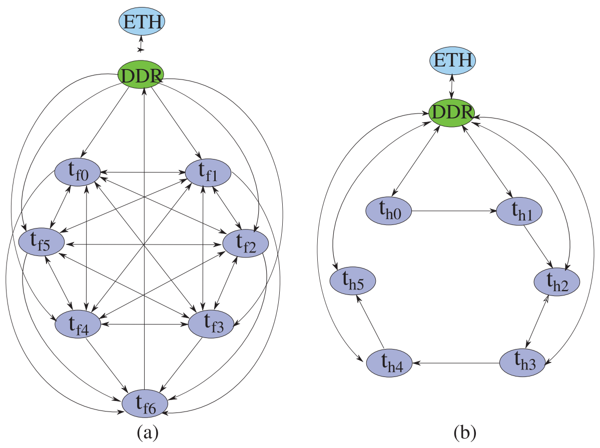


Figure 4: Task graph of core-to-core and core-to-I/O communications of the: (a) FADEC application, (b) HM application.

FADEC applications are critical, while HM applications are non-critical. In this case study, we consider 2 configurations: one composed of 8 applications (3 FADEC and 5 HM) and one composed of 9 applications (3 FADEC and 6 HM). BAGs of the sent data by these applications are indicated in Table 4. HM_7 is the added application between the two configurations.

5.2 Mapping applications into the manycore

The described applications are the mapped into the many-core. Different strategies can be used such as:

- **Smart Hill Climbing (SHiC)** [11]: this approach maps the applications without fragmented regions, generated by the methods in the literature (as in [9], [19],[12]). This method searches a region of size equal to the size of the application to be allocated. The tasks of this application are allocated in the selected region in such a way to reduce the distance between the communicated tasks. Thus, SHiC allocates the task with the maximum number of communications at the center of this region and around it the tasks communicating with it to form a square shape.
- **$\mathbf{Map_{IO}}$** [4]: this approach performs the mapping into two steps. The first step splits the NoC into regions and then allocates primarily critical applications in a dedicated region close to memory and Ethernet controllers by following a circular direction and using rectangular shapes. The second step consists to allocate the tasks within each application where some rules are used to minimize the contentions on the path of the core-to-I/O flows. These rules are based on the principle of allocating the tasks which generate perpendicular communications on the path of the core-to-I/O flows.
- **$\mathbf{ex\_Map_{IO}}$** [2]: this approach is an extension of Map_{IO} in order to minimize the delay of outgoing I/O flows on the NoC (such as d, d1 and d2 in Figure 3). Several mapping rules have been defined. As an example, one rule minimizes the number of flows that can delay an outgoing flow on its path to the Ethernet interface.

As an example, Figure 5 shows the mapping of the two configurations (8 applications and 9 applications) on a 10x10 NoC using ex_Map_{IO}.

The number of occupied cores is 89 for an 8 applications configuration and 96 for a 9 applications configuration. This means that there are 11 free cores for the first configuration and 4 in the other one, as we can see in Figure 5. These free cores can execute a t_{DDR}. As an example, t_{DDR} can be located in column 2 of line 4 in Figure 5a and can be located in column 3 of line 3 in Figure 5b.

Once the applications are mapped, we can compute the WCTT of the transmission of the VLs from the memory to the Ethernet interface. The results for the two considered configurations are given in Table 4. In this case study, we consider that for each application a maximum size frame is sent through the AFDX network. So, the frame transmission duration is 123 μs for all the applications of the case study. The number of slots allocated for each application can be computed and is given in Table 4, by considering Equation **??**. Note that there are no value for SHiC considering 9 applications since mapping 9 applications using SHiC is not possible.

5.3 Obtained scheduling table

Figure 6 shows the resulting scheduling tables for the different mapping approaches. As explain in section 4, the number of lines in which all the applications have to be allocated is 2. This is why we only represent the 2 first lines of the scheduling tables in Figure 6. The oversampling of slots is then 2ms. This means that applications with a BAG greater than 2ms will get a transmission slot every 2ms. If they miss the first transmission slot, they will have to wait a maximum of 2ms before the data will be transmitted.

5.4 Results

The results are shown in Table 5. Distributed t_{DDR} corresponds to the solution described in Section 3. In this case, SHiC can not respect the jitter constraint while Map_{IO} and ex_Map_{IO} allows a reduction of the jitter leading to a jitter value less than 500μs for an 8 applications mapping. But, the jitter is still greater than 500μs when mapping 9 applications.

As expected, the solution proposed in the paper significantly reduces the jitter of the VLs. The jitter becomes lower than 200μs when mapping both 8 and 9 applications on the NoC for all the mapping strategies.

6 CONCLUSION

In this paper, we proposed to replace the mono-core processors in avionics architecture by a NoC-based many-cores architecture. Thus, End Systems are replaced by many-cores.

The main contribution of the paper is that it proposes a new VL transmission strategy which considers one dedicated node in the many-core architecture to shape the traffic and schedules the outgoing I/O flows. This dedicated node executes a scheduling table. The applications are allocated into the table using an ILP formulation. The two objectives of the strategy is to guarantee the BAG regulation and to oversample the slots to applications with a BAG greater than 2ms.

The results show that the jitter is significantly reduced and only depends on the interferences that can occur on the transmission path between the DDR and the I/O interface.

The ILP formulation imposes a strong constraint that can be relaxed: all the applications should be stored in the minimum period (different to 1ms). As an example, we can notice that it remains a very little amount of free slots in Figures 6d and 6e (respectively 3 and 6). But applications with longer BAG can be allocated after this minimum period. This leads to consider variable capacity size and to formulate a variant of bin packing [14] or a cutting stock problem with periodic items and thus defining a more complex strategy for filling the scheduling table.

The worst case transmission delay of a VL from one application from a many-core to another application executed on another many-core should be mastered in order to certify the avionic system. AFDX network uses Network Calculus to compute the maximum worst case delay [6]. For a NoC used in processors like Tilera, the method is Recursive Calculus [5]. The problem that could be addressed now is how to compute the worst case end-to-end delay of the global communication.

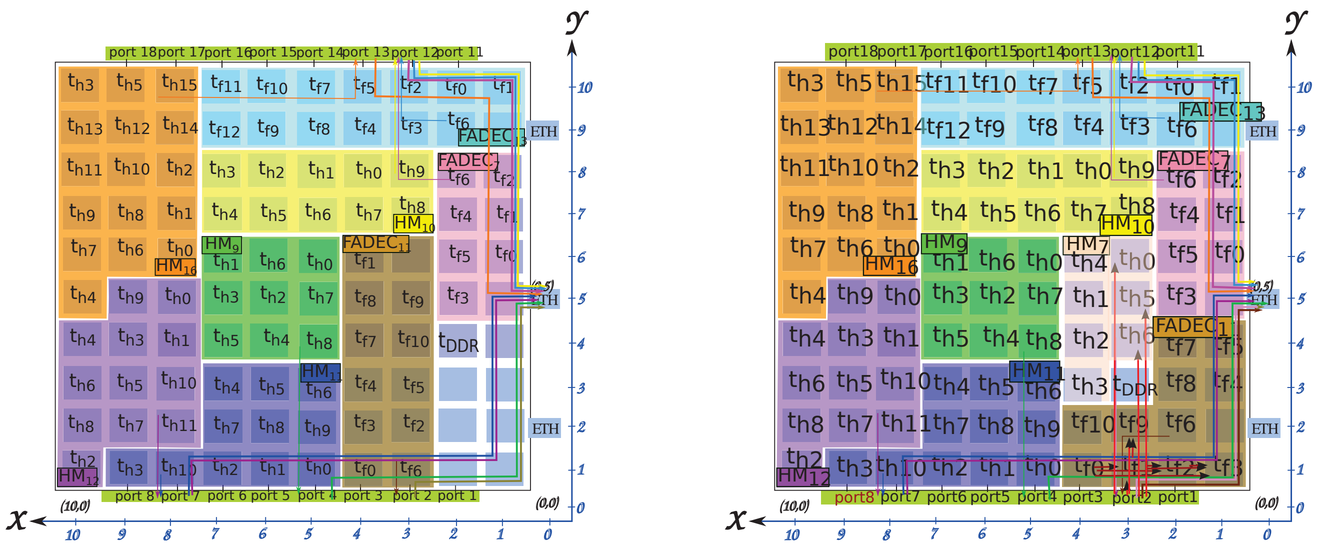


Figure 5: Mapping 8 (left) and 9 (right) applications on a 10x10 many-core using *ex_Map$_{IO}$*.

		8 applications						9 applications			
		SHiC		Map$_{IO}$		ex_Map$_{IO}$		Map$_{IO}$		ex_Map$_{IO}$	
Applications	BAG	WCTT	# slots	WCTT	# slots	WCTT	# slots	WCTT	# slots	WCTT	# slots
FADEC$_7$	4	107	8	52	6	51	6	52	6	51	6
FADEC$_{11}$	8	183	10	117	8	77	7	105	8	68	7
FADEC$_{13}$	16	111	8	52	6	51	6	52	6	51	6
HM$_7$	4	-	-	-	-	-	-	105	8	68	7
HM$_9$	2	107	8	17	5	16	5	37	6	33	5
HM$_{10}$	16	18	5	52	6	51	6	52	6	51	6
HM$_{11}$	32	20	5	23	5	17	5	43	6	34	6
HM$_{12}$	16	20	5	23	5	17	5	43	6	34	6
HM$_{16}$	32	23	5	157	9	139	9	157	9	139	9

Table 4: WCTT on the NoC of the transmission of VL flows of each application from the memory to the Ethernet interface, (in μs)

	8 applications			9 applications	
	SHiC	Map$_{IO}$	ex_Map$_{IO}$	Map$_{IO}$	ex_Map$_{IO}$
distributed t$_{DDR}$	590.5	492.9	419.5	645.9	528.9
one t$_{DDR}$	183	157	139	157	139

Table 5: Maximum jitter experienced by the transmission of VL, in μs

REFERENCES

[1] Aeronautical Radio Inc. ARINC 664. 2005. *Aircraft Data Network, Part 7: Avionic Full Duplex Switched Ethernet (AFDX) Network.*

[2] Laure Abdallah, Jérôme Ermont, Jean-Luc Scharbarg, and Christian Fraboul. 2017. Towards a mixed NoC/AFDX architecture for avionics applications. In *IEEE 13th International Workshop on Factory Communication Systems, WFCS 2017, Trondheim, Norway, May 31 - June 2, 2017.* 1–10. https://doi.org/10.1109/WFCS.2017.7991950

[3] Laure Abdallah, Mathieu Jan, Jérôme Ermont, and Christian Fraboul. 2015. Wormhole networks properties and their use for optimizing worst case delay analysis of many-cores. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES).* Siegen, Germany, 59–68.

[4] Laure Abdallah, Mathieu Jan, Jérôme Ermont, and Christian Fraboul. 2016. Reducing the contention experienced by real-time core-to-I/O flows over a Tilera-like Network on Chip. In *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on.* IEEE, 86–96.

[5] H. Ayed, J. Ermont, J. l. Scharbarg, and C. Fraboul. 2016. Towards a unified approach for worst-case analysis of Tilera-like and KalRay-like NoC architectures. In *2016 IEEE World Conference on Factory Communication Systems (WFCS).* 1–4.

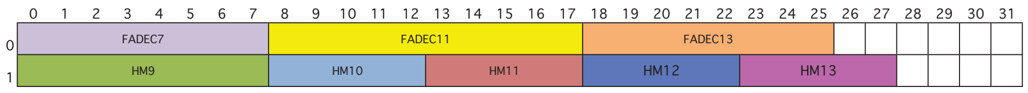
[6] H. Charara, J. Scharbarg, J. Ermont, and C. Fraboul. 2006. Methods for bounding end-to-end delays on an AFDX network. In *Proc of the 18th Euromicro Conf. on Real-Time Systems (ECRTS).* Dresde, Germany, 193–202.

[7] IBM ILOG CPLEX. 2009. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.

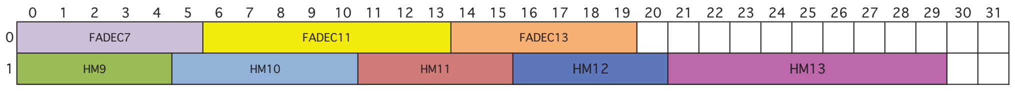
[8] Benoît Dupont de Dinechin, Duco van Amstel, Marc Poulhiès, and Guillaume Lager. 2014. Time-critical Computing on a Single-chip Massively Parallel Processor. In *Proc. of the Conf. on Design, Automation & Test in Europe (DATE'14).* 97:1–97:6.

[9] Ewerson Luiz de Souza Carvalho, Ney Laert Vilar Calazans, and Fernando Gehm Moraes. 2010. Dynamic task mapping for MPSoCs. *Design & Test of Computers* 27, 5 (2010), 26–35.

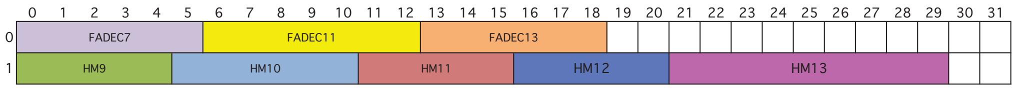
[10] RTCA DO. 2011. 178C. *Software considerations in airborne systems and equipment certification* (2011).



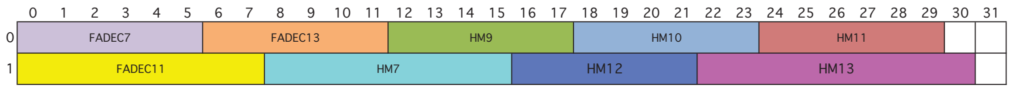
(a) SHiC



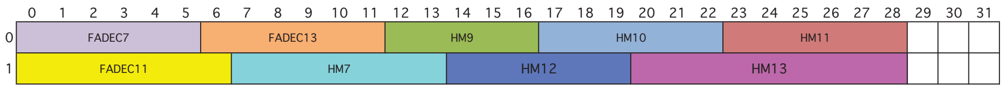
(b) Map$_{IO}$ with 8 applications



(c) Ex_Map$_{IO}$ with 8 applications



(d) Map$_{IO}$ with 9 applications



(e) Ex_Map$_{IO}$ with 9 applications

Figure 6: Two first lines of the resulting scheduling tables

[11] Mohammad Fattah, Masoud Daneshtalab, Pasi Liljeberg, and Juha Plosila. 2013. Smart hill climbing for agile dynamic mapping in many-core systems. In *Proc. of the 50th Annual Design Automation Conference*. 39.

[12] Mohamamd Fattah, Marco Ramirez, Masoud Daneshtalab, Pasi Liljeberg, and Juha Plosila. 2012. CoNA: Dynamic application mapping for congestion reduction in many-core systems. In *30th Intl. Conf. on Computer Design (ICCD)*. 364–370.

[13] Thomas Ferrandiz, Fabrice Frances, and Christian Fraboul. 2011. Using Network Calculus to compute end-to-end delays in SpaceWire networks. *SIGBED Review* 8, 3 (2011), 44–47.

[14] Jangha Kang and Sungsoo Park. 2003. Algorithms for the variable sized bin packing problem. *European Journal of Operational Research* 147, 2 (2003), 365–372.

[15] Silvano Martello and Paolo Toth. 1990. Knapsack Problems, J.

[16] Prasant Mohapatra. 1998. Wormhole routing techniques for directly connected muti-computer systems. *ACM Computer Survey (CSUR)* 30, 3 (September 1998), 374–410.

[17] Vincent Nélis, Patrick Meumeu Yomsi, Luís Miguel Pinho, José Carlos Fonseca, Marko Bertogna, Eduardo Quiñones, Roberto Vargas, and Andrea Marongiu. 2014. The Challenge of Time-Predictability in Modern Many-Core Architectures. In *14th Intl. Workshop on Worst-Case Execution Time Analysis*. Madridr, Spain, 63–72.

[18] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. 2007. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro* 27, 5 (2007), 15–31.

[19] Christopher Zimmer and Frank Mueller. 2012. Low contention mapping of real-time tasks onto tilepro 64 core processors. In *18th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 131–140.