Limiting over sampling to improve transmission schedulability in a mixed NoC/AFDX architecture

Sandrine Mouysset, Jérôme Ermont, Jean-Luc Scharbarg

Limiting over sampling to improve transmission schedulability in a mixed NoC/AFDX architecture

Sandrine Mouysset
Université de Toulouse - IRIT - UPS
Toulouse, France

Jérôme Ermont, Jean-Luc Scharbarg
Université de Toulouse - IRIT - INPT/ENSEEIHT
Toulouse, France

***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 500μs. 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 500μs. A first solution has been proposed, where a single task in each many-core manages the transmission of the VLs. This task executes a scheduling table. The access to the Ethernet interface is then only allowed to one VL leading to a significant reduction of the jitter. By oversampling the VL transmissions in a minimum period, the waiting delays are also reduced. But this solution limits the number of VLs. In this paper, we propose to improve the transmission scheduling by relaxing constraint on the over sampling. A new scheduling table is constructed using an Integer Linear Program. This solution increases the number of VLs transmitted by the many-core and still reduces the waiting delays for the transmission of the VLs.**

I. Introduction

Aircrafts are equipped with numerous electronic equipments. 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) [1]. Such an architecture is depicted in Figure 1. Computing systems are interconnected by an AFDX (Avionics Full Duplex Switched Ethernet) [2]. The End System (ES) provides an interface between a processing unit and the 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 [3]. 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 [4], while Kalray MPPA has 8 Ethernet interfaces and 2 memory controllers [5].

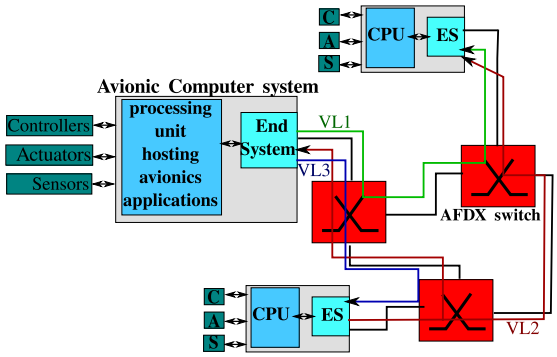


Fig. 1: An AFDX network.

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 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. [6] 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 [7], the second factor is addressed. The authors propose a static scheduling of Ethernet transmissions, based on a table. Each transmission is allocated in 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.

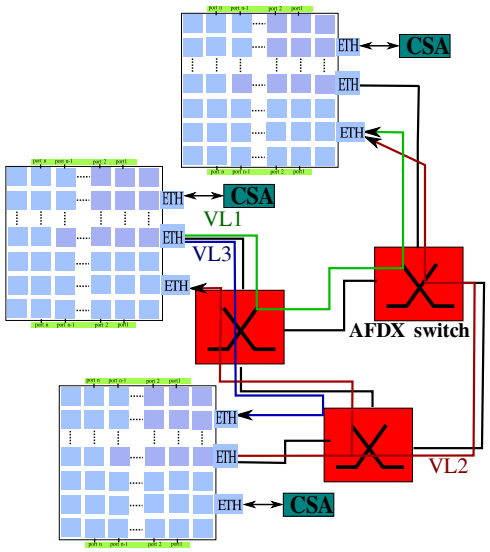


Fig. 2: A mixed NoC/AFDX architecture

The two objectives of the allocation strategy is to guarantee the BAG regulation and to over sample the slots to applications with a BAG greater than 2ms. 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.

In this paper, we propose to extend the approach of [7]. The goal is to relax the strong constraint of the method: all the applications should be stored in a minimum period (different to 1ms). But applications that send VLs with longer BAG can be allocated after this minimum period. So, we propose a new allocation of the slots to the applications formulated 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 over sample the slots for outgoing flows in order to reduce the waiting delays before the transmission slots.

The remainder of the paper is as follows. Section II introduces current AFDX and NoC architectures. Section III proposes an illustrative avionics case study. Section IV explains how a VL can be transmitted using a dedicated node which executes a scheduling table, as it has been defined in [7]. The new approach is described in Section V. Section VI shows the proposed scheduling solution on the case study and give some results. Finally, Section VII concludes with some future works.

II. System model

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

A. 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.

B. 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 [8], 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 [9], [10]. In this paper, we choose RC_{NoC} [11] 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.

III. Illustrative case study

An avionics case study is proposed in order to illustrate the problem and our solution. It is composed of critical and non-critical applications that are mapped into the manycore.

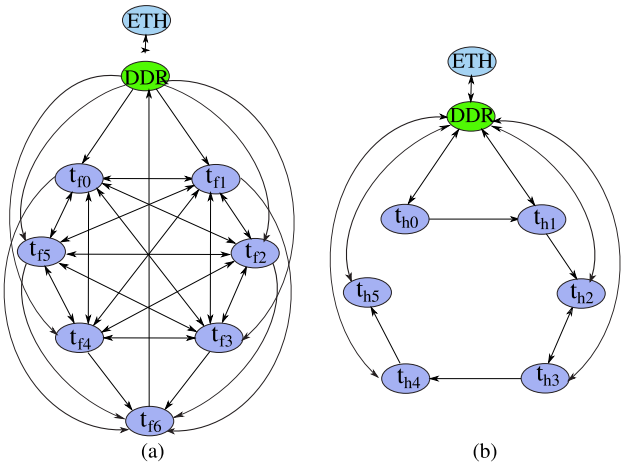


Fig. 3: Task graph of core-to-core and core-to-I/O communications of the: (a) FADEC application, (b) HM application.

A. Considered applications

The considered case study is composed of 2 types of 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 2×1500 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 3a 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 2×1500 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 3b shows the tasks graph of HM$_6$.

FADEC applications are critical, while HM applications are non-critical. These applications will send 2 Vls each through the Ethernet interface. The case study is composed of 3 FADEC applications and 6 HM applications. The configuration of the VL sent by these applications is given in Table I. These applications are mapped into the manycore.

B. Mapping the applications into the manycore

The described applications are then mapped into the manycore. Different strategies can be used such as:

- **Smart Hill Climbing (SHiC)** [12]: this approach maps the applications without fragmented regions, generated by the methods in the literature (as in [13], [14],[15]). 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.
- **Map$_{IO}$** [16]: 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.
- **ex_Map$_{IO}$** [6]: this approach extends Map$_{IO}$ in order to minimize the delay of outgoing I/O flows on the NoC.

Applications	VL	BAG	Applications	VL	BAG
FADEC$_7$	VL1 VL2	4 32	HM$_{10}$	VL11 VL12	16 64
FADEC$_{11}$	VL3 VL4	8 16	HM$_{11}$	VL13 VL14	32 4
FADEC$_{13}$	VL5 VL6	16 32	HM$_{12}$	VL15 VL16	16 64
HM$_7$	VL7 VL8	4 16	HM$_{16}$	VL17 VL18	4 32
HM$_9$	VL9 VL10	2 2			

TABLE I: VL configurations per application

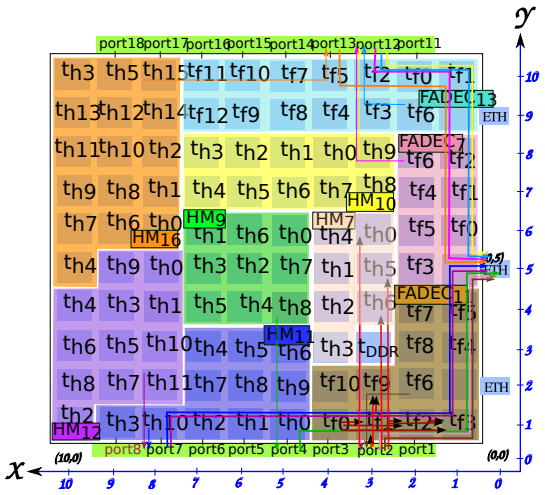


Fig. 4: Mapping 9 applications on a 10x10 many-core using ex_Map$_{IO}$.

Figure 4 shows an example of mapping for configurations of 9 applications on a 10x10 NoC using ex_Map$_{IO}$. The number of occupied cores is 96 for this configuration. This means that there are 4 free cores as we can see in Figure 4. These free cores can be dedicated to schedule the transmission of outgoing I/O flows.

IV. Outgoing flows transmission

To avoid that the Ethernet interface is busy when it receives a DMA command (like for VL$_1$ second period in Figure 5), a dedicated core is used to transmit the outgoing I/O flows. These flows are scheduled by the execution of a table.

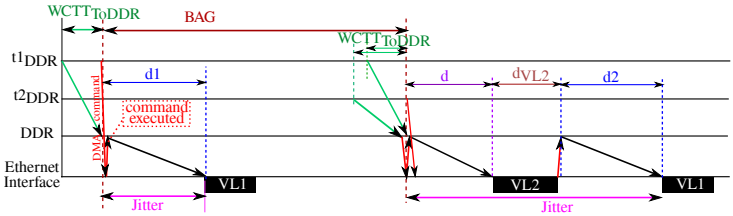


Fig. 5: A possible transmission on a given VL.

A. A dedicated core for outgoing I/O flows

On many-core architecture, 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 5. Two VLs VL1 and VL2 are respectively generated by tasks $t1_{DDR}$ and $t2_{DDR}$. At the beginning of VL1 first BAG period, $t1_{DDR}$ sends VL1 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 bounded 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 VL1 first BAG period in Figure 3, the jitter is the delay $d1$ of this transmission, which is between 0 and its worst-case duration. Indeed, for VL1 second BAG period, the Ethernet interface is busy with VL2 when it receives VL1 DMA command. The delay due to VL2 has to be added to the jitter, leading to an overall value of $d+d_{VL2}+d2$.

As in [7], a specific core of the NoC is dedicated to the transmission of VL through the Ethernet interface. The goal is to avoid that the Ethernet interface is busy when it receives a DMA command (like for VL1 second period in Figure 5). The node is chosen between the ones that do not execute any tasks. As an example, node located located in column 3 of line 3 in Figure 4 can be chosen. On this node, the task t_{DDR} executes the transmission commands from the DDR to the Ethernet interface. The transmission of these commands uses another internal network and does not impact the communications of other tasks. The transmission of the corresponding data are then transmitted through the NoC from the DDR to the Ethernet interface.

In order to reduce the jitter due to the transmission on the NoC, the task t_{DDR} executes a scheduling table. Thus, only one VL is sent on the NoC at a given time. Then the only jitter that appends is due to the transmission of the VL from the DDR to the Ethernet interface though the NoC (as for VL1 first period in Figure 5).

For the considered case study described in Section III, the WCTT of the VL transmitted through the NoC is given in Table II. This delay is computed using the RC_{NoC} method [11] and is the maximum bounded delay that to transmit a VL from the DDR to the Ethernet interface. Note that the WCTT of all the VLs sent by a same application is the equal since they use the same path on the NoC and have the same frame size (1500 bytes). They only differ in BAG.

B. A scheduling table

In order 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, a scheduling table is constructed. The considered scheduling table is composed of slots of 31.25μs. The global duration of the table is 128ms. So the number of slots is 4096. The table is composed of 128 lines of 1ms, 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 table is represented in Figure 6. A first allocation solution has been defined with two objectives:

1) The allocation of the slots in order to guarantee that the VLs sent by the applications respect their BAG constraints, *i.e.* which 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 current transmission.
2) Suppose that a VL of an application can be sent at column 20 in line 15 in the table. If this VL is ready to be sent at column 2 in line 1, its transmission will start 15.5 ms later (15 lines of the table is 15 ms, 16 slots of 31.25μs is 0.5 ms). To reduce this waiting delay, more slots are reserved for the transmission of the VLs. This oversampling is fixed to the minimum value of the BAGs of the VLs except 1ms.

For each application which sends a VLi, the number of allocated slots ω_i for VLi is defined as:

$$\omega_i = \left\lceil \frac{\text{WCTT(VLi)} + \text{ft(VLi)}}{\text{sd}} \right\rceil \quad (1)$$

where WCTT(VLi) is the WCTT of VLi from the DDR to the Ethernet interface, ft(VLi) is the transmission delay of the frame through the Ethernet interface and $\text{sd} = 31.25\mu s$ is the slot duration. The WCTT(VLi) depends on the contention on the NoC, while ft(VLi) depends on the size of the frame that is transmitted by the Ethernet interface. For example, VL1 which is sent by $FADEC_7$, as defined in Table I. Using ex_Map_{IO} mapping method, the WCTT of this VL from the DDR to the Ethernet interface is WCTT(VL1)=51μs. At 100 Mbps, the transmission of this VL through the Ethernet interface takes ft(VL1)=$\frac{1500\times8}{100.10^6}$=120$\mu$s. So the resulting number of slots allocated to VL1 is $\omega_1 = \left\lceil \frac{51+120}{31.2} \right\rceil = 6$. Table II gives the number of slots allocated to the VLs for all the applications, by considering the different mapping methods.

	8 applications						9 applications			
	SHiC		Map_{IO}		ex_Map_{IO}		Map_{IO}		ex_Map_{IO}	
Applications	WCTT	# slots	WCTT	# slots	WCTT	# slots	WCTT	# slots	WCTT	# slots
$FADEC_7$	107	8	52	6	51	6	52	6	51	6
$FADEC_{11}$	183	10	117	8	77	7	105	8	68	7
$FADEC_{13}$	111	8	52	6	51	6	52	6	51	6
HM_7	-	-	-	-	-	-	105	8	68	7
HM_9	107	8	17	5	16	5	37	6	33	5
HM_{10}	18	5	52	6	51	6	52	6	51	6
HM_{11}	20	5	23	5	17	5	43	6	34	6
HM_{12}	20	5	23	5	17	5	43	6	34	6
HM_{16}	23	5	157	9	139	9	157	9	139	9

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

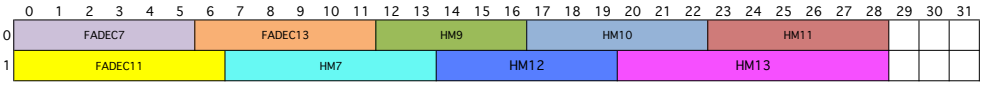


Fig. 6: Scheduling table of a reduced version of the considered case study using ex_Mapio_{IO}

.

An example of scheduling table is presented in Figure 6. It is obtained by using a bin packing method. This example is a reduced version of the illustrative case study. It is composed of VL1, VL3, VL5, VL7, VL9, VL11, VL13, VL15 and VL17 (9 VLs are transmitted). As the minimum BAG value is 2ms (for VL9), the slots are packed in two lines of the table. 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 their first transmission slot, they will have to wait a maximum of 2ms before the data will be transmitted.

Suppose that we want to add VL8 in Figure 6. As it remains a very little amount of free slots (3 slots) when the 9 previous VLs slots are allocated, VL8 cannot be transmitted by the I/O core. However, the BAGs of VL7 and VL8 are 4 ms. If the constraint of the minimum BAG is relaxed, VL7 can be stored in slots of line 1 and VL8 can be stored in slots of line 3, as in Figure 7d. The BAG constraint for these VLs is still guaranteed by repeating the 4 first lines of the table. A new allocation approach should be defined in order to add more VLs, and thus to transmit all the VLs of the case study proposed in Section III, by relaxing the strong constraint of the over sampling.

V. Our approach for extending the number of transmitted VLs

In this paper, we propose to relax the strong constraint of the minimum BAG value of all the VLs that have to be transmitted. The idea is to determine first the minimum number of lines in which all the slots of the VLs can be stored. The allocation is then done in 2 steps:

1) finding a first allocation of the slots into this obtained number of lines. This allocation guarantees the respect of the BAG but limits the over sampling to the period associated to this number of lines;
2) allocating more slots to the VLs with a BAG greater than the period connected to the obtained number of lines if there is enough free slots after the execution of step 1. This step allows to increase the possible over sampling.

A. Minimum BAG to allocate all the VLs

For allocating all the VLs within an optimal period, we can estimate the value of this period, noted N, by taking into account both BAG and capacity of each VL compared with the capacity of each line of the scheduling table. This leads to define a BAG among the different possible BAG values within the set $\{2^1, .., 2^7\}$ and the following positive objective function:

$$N = \arg \min_{i \in \{2^1,..,2^7\}} i \times C - \left(\sum_{j \in P} \frac{i}{BAG_{VL_j}} \omega_j + \sum_{k \in L} \omega_k \right) \quad (2)$$

where $P = \{VL_j | BAG_j \leq i\}$ and $L = \{VL_k | BAG_k > i\}$ are the sets of VLs which BAG is less (respectively greater) than i and C is the capacity of one line of the scheduling table. In this equation, the first term $i \times C$ represents the total capacity from the i first lines in the table and the second term represents the total capacity occupied by the VLs and their respective periods if all the VLs are allocated within these lines. So negative values will indicate that the configuration cannot ensure the transmission, *i.e.* there is not enough capacity to transmit all the VLs. The result should be positive for estimating a possible minimum BAG to allocate all the VLs. This value N represents the minimum number of lines considered for VL allocation.

B. VLs Allocation

With the notations in the Table III, we consider m different possible VLs and their associated BAG. We compute the number of times the different VLs should be stored in the N lines of the scheduling table as follows :

$$n_j = \frac{N}{BAG_j}, \forall j \in \{1, .., m\}, \quad (3)$$

Constants	
m	Number of VLs
C	Full capacity of one line in the scheduling table
N	Minimum number of lines considered for allocation
BAG_j	period of the VL j noted VL_j
ω_j	number of slots for VL j
n_j	number of times VL_j appears in N lines with respect to its BAG_j
C_i	Free slots of each line i after VLs 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 VL 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 VLs

TABLE III: ILP Notations for VL allocation Problem

with N defined by Equation (2). When the number of VLs with same BAGs increases, the constraint of the integer capacity of each table line has to be taken into account for allocation. That's why the allocation of each VL can be formulated as an Integer Linear Problem (ILP). This problem can be considered as a variant of bin packing formulation with an additional constraint on the periodicity of the VLs. It can be formulated as follows : by considering m VLs, 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 VLs into the minimum number of lines in the table so that the total number of slots per VL packed in any line of the table does not exceed its capacity and it satisfies the periodicity of each VL. This will provide an optimal way to assign slots in the scheduling table by the VLs. This periodicity constraint imposes that some VL should appear several times in the number of lines N in the table given by equation (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 VL j in the line i of the table as defined in the table III. From these notations, as the objective is to pack all the VLs 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 III. This function has to be minimized so that the total number of slots per VL, 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 VLs and their periodicity, this leads to define these following periodicity constraints:

$$x_{ij} = x_{i+BAG_j j}, \forall j = 1,..,m, \forall i = 1,..,N,$$

and that the VLs should be exactly assigned n_j times in the table as :

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

where n_j is defined by equation (3).

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

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

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

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

$$x_{ij} \quad = x_{i+BAG_j j}, \forall j = 1,..,m, \forall i = 1,..,N \tag{7}$$

$$y_i \in \{0,1\} \quad , \forall i = 1,..,N \tag{8}$$

$$x_{ij} \in \{0,1\} \quad , \forall i = 1,..,N, \forall j = 1,..,m. \tag{9}$$

Objective function (4) means that the VLs should be stored in a minimum of lines of table N. Constraints (5) impose that the capacity C of each line is not exceeded while constraints (6) and (7) ensure that all the periodicity properties of the VLs are satisfied. This ILP problem is solved by using CPLEX [17].

C. VL over sampling

From this first storage of the different VLs, the scheduling table provides free slots that can be addressed for other VLs. So we consider now the filling of the table so that it permits transmitting applications with long BAG several times within its own BAG. The principle is the same as the previous one but the capacity is no more constant but variable. Let consider the capacity, noted C_i, as the free slots for each line i of the scheduling table defined by :

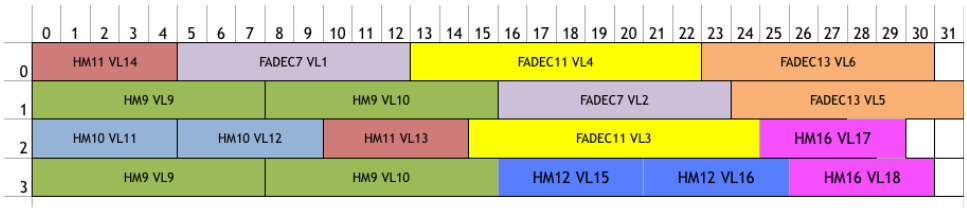
$$C_i = C - \sum_{j \in S} \omega_j, \tag{10}$$

where $S = \{j | y_j = i\}$ gathers all the VLs that are assigned at the line i. The capacity C_i should be in $\{0,..,C\}$.

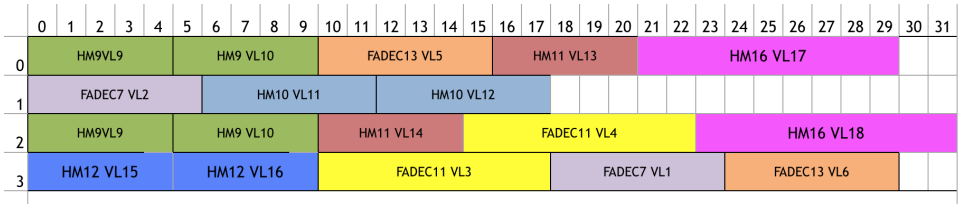
To do the filling of free slots, we recursively solve the same previous ILP problem defined by equations (4)-(6)-(7)-(8)-(9) but the capacity constraint should include variable capacity for each line. This leads to replace the equation (5) by the following one :

$$\sum_{j=1}^{m} \omega_j x_{ij} \leq C_i\ y_i, \forall i = 1,..,N \tag{11}$$

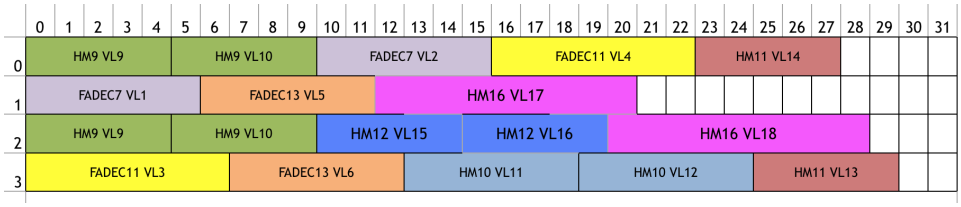
where C_i is defined by equation (10). The VLs with long BAGs (for examples, 32ms, 64ms, 128ms) are first privileged for oversampling to ensure an earlier transmission. After allocating these VLs, if there is still some free slots that can satisfy all the constraints of periodicity, VLs with short BAGs (for examples, 8ms, 16ms) can be considered for allocation.



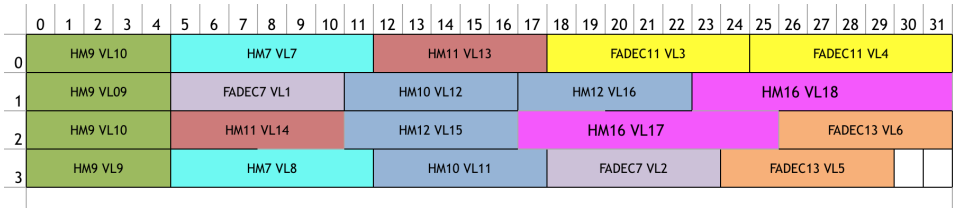
(a) SHiC with 8 applications



(b) Map_{IO} with 8 applications



(c) exMap_{IO} with 8 applications



(d) exMap_{IO} with 9 applications

Fig. 7: Resulting scheduling tables for SHiC, Map_{IO} and exMap_{IO} with 8 applications and for exMap_{IO} with 9 applications.

VI. Results on the Case Study

In this section, we evaluate the approach to the illustrative case study.

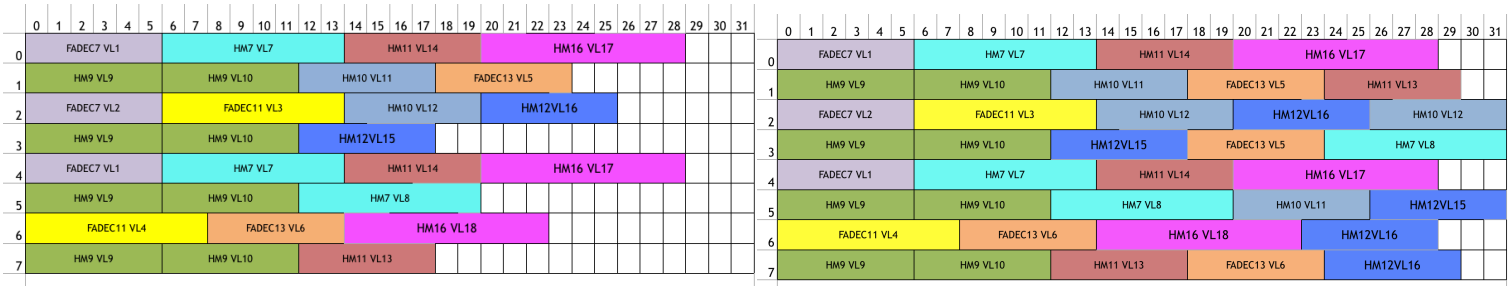
Figure 7 shows the resulting scheduling tables for SHiC, Map_{IO} and exMap_{IO} with 8 applications and for exMap_{IO} with 9 applications. The computation time of the ILP system is between 20 ms and 30 ms for all the configurations. First, the minimum number of lines is computed using Equation (2) of Section V-A. All the VLs can be allocated in 4ms, *i.e.* 4 lines of the table. This is why we only represent the 4 first lines of the scheduling tables in Figure 7. The over sampling of slots is then 4 ms. This means that VLs with a BAG greater than 4ms will get a transmission slot every 4ms. If they miss the first transmission slot, they will have to wait a maximum of 4ms before the data will be transmitted.

For Map_{IO} with 9 applications, Equation (2) indicates that it needs 8 lines to allocate all the VLs. As said before, this means that VLs with a BAG greater than 8ms will get a transmission slot every 8ms. A first allocation is then constructed and is given in Figure 8a. For this first allocation, the proposed over sampling provides free slots that can be allocated for VLs with BAG greater than 8ms. A second step is then executed in order to full fill the remaining free slots. The resulting table is depicted in Figure 8b. VLs with BAG greater than 8ms, such as VLs from HM_{12}, are then allocated another time in the table. Thus, their waiting delay is reduced. As an example, in Figure 8a, if VL16 is ready to be sent after slot 26 of line 2 (it misses its first slot), it will wait a maximum duration of 8ms before its transmission through the Ethernet interface. This VL will wait a maximum of around 4ms in Figure 8b.

Adding more VLs will limit more the over sampling. The particular case where N of Equation (2) is equal to the maximum BAG value of the VLs do not guarantee the allocation of the VLs. If the VLs allocation of Section V-B gives a solution, all the VLs will respect their BAG constraint but the over sampling is limited to the remaining capacity size in the table.

VII. Conclusion

In this paper, we proposed to replace the mono-core processors in avionics architecture by a NoC-based many-cores



(a) Map_{IO} with 9 applications, first step

(b) Map_{IO} with 9 applications, second step

Fig. 8: Resulting scheduling table for Map_{IO} with 9 applications.

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 over sample the slots to VLs in order to reduce the waiting delays of the transmission of the VLs. The idea is to determine the minimum number of lines needed in which the transmission slots of all the VLs can be stored. To maximize the over sample, the remaining free slots are allocated to VLs with a BAG greater than this obtained minimum value. The proposed solution thus limits the over sampling and allows to transmit more VLs than the solution proposed in [7].

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 [18]. For a NoC used in processors like Tilera, the method is Recursive Calculus [10]. The problem that could be addressed now is how to compute the worst case end-to-end delay of the global communication.

References

[1] DO-RTCA, "178c," *Software considerations in airborne systems and equipment certification*, 2011.

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

[3] V. Nélis, P. M. Yomsi, L. M. Pinho, J. C. Fonseca, M. Bertogna, E. Quiñones, R. Vargas, and A. Marongiu, "The Challenge of Time-Predictability in Modern Many-Core Architectures," in *14th Intl. Workshop on Worst-Case Execution Time Analysis*, Madrid, Spain, 2014, pp. 63–72.

[4] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[5] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Proc. of the Conf. on Design, Automation & Test in Europe (DATE'14)*, 2014, pp. 97:1–97:6.

[6] L. Abdallah, J. Ermont, J. Scharbarg, and C. Fraboul, "Towards a mixed NoC/AFDX architecture for avionics applications," in *IEEE 13th International Workshop on Factory Communication Systems, WFCS*, 2017, pp. 1–10.

[7] J. Ermont, S. Mouysset, J. Scharbarg, and C. Fraboul, "Message scheduling to reduce AFDX jitter in a mixed NoC/AFDX architecture," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS*, 2018, pp. 234–242.

[8] P. Mohapatra, "Wormhole routing techniques for directly connected muti-computer systems," *ACM Computer Survey (CSUR)*, vol. 30, no. 3, pp. 374–410, September 1998.

[9] T. Ferrandiz, F. Frances, and C. Fraboul, "Using Network Calculus to compute end-to-end delays in SpaceWire networks," *SIGBED Review*, vol. 8, no. 3, pp. 44–47, 2011.

[10] H. Ayed, J. Ermont, J. l. Scharbarg, and C. Fraboul, "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)*, May 2016, pp. 1–4.

[11] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "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, June 2015, pp. 59–68.

[12] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proc. of the 50th Annual Design Automation Conference*, 2013, p. 39.

[13] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for mpsocs," *Design & Test of Computers*, vol. 27, no. 5, pp. 26–35, 2010.

[14] C. Zimmer and F. Mueller, "Low contention mapping of real-time tasks onto tilepro 64 core processors," in *18th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2012, pp. 131–140.

[15] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Cona: Dynamic application mapping for congestion reduction in many-core systems," in *30th Intl. Conf. on Computer Design (ICCD)*, 2012, pp. 364–370.

[16] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul, "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, 2016, pp. 86–96.

[17] CPLEX, IBM ILOG, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[18] H. Charara, J. Scharbarg, J. Ermont, and C. Fraboul, "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, July 2006, pp. 193–202.