Praise Worthy Prize

International Review on Computers and Software (I.RE.CO.S.), Vol. 7, N. 2
ISSN 1828-6003 *March 2012*

Modelling and Evaluating a Crossbar Switched Fabric CAN Network Using Stochastic and Colored Petri Nets

Mohamed Mazouzi, Oussama Kallel, Salem Hasnaoui, Mohamed Abid

Abstract – *In recent years, the demand for sophisticated embedded systems requires the use of many connected equipments. CAN buses were developed for connecting many microcontrollers which oversee many Electronic Control Units (ECU). Because of its no deterministic performance and its limited bandwidths and throughput, existing CAN-Bus has presented some problems. Switched Fabric CAN Network can be a fast and reliable hardware solution. In fact, high performance, reliability and predictability require crossbar switched fabric network.*
In this paper, we proposed a switched fabric CAN network Architecture based on CAN Controllers and switched fabric. This network is modeled and verified by the use of timed colored Petri nets. To extract network performance metrics, the simulation of the whole model was done using CPNTools.

Keywords: *Embedded Systems, ECU, CAN Bus, Switched Fabric CAN Network, Network Performance Evaluation, Timed Colored Petri Nets, Cpntools*

I. The Switched Fabric CAN Network Architecture

The limitations of a bussed network [1] are eliminated with crossbar switch network. A switched-fabric bus is unique in that it allows all CAN Controllers on a bus to logically interconnect with all CAN Controllers on the bus.

The switching fabric is the physical connection within a switch between the input and output ports; it can be proved that all switches need a crossbar inside their switching fabric which allow them to operate at very high speed. Crossbar switches are widely used because of their simplicity and their high-performances [2] which promise to greatly simplify efforts and to add better capability and availability.

Crossbar switch [3] can support simultaneously multiple messages. This greatly increases the aggregate bandwidth of the system. Because of the broadcast nature of the CAN [4] protocol (ie: messages are not sent to a specific destination address, but rather as a broadcast).The chosen crossbar switch (as it is shown in Figure 1) is configured by closing all its crosspoints to ensure that the CAN message will be sent at the same time [5],[6] for all outputs nodes as it is defined in CAN protocol [7].

CAN messages need to be queued in buffers when short-term overloading occurs, where the sum of input rates for a single output port exceeds the outgoing link rate. Buffering (queueing) characterizes all kinds of switches.

We can use all types of queueing architectures: “Output Queueing (OQ)”, “Input Queueing (IQ)”, “and Combined Input Output Queueing (CIOQ - Internal Speedup), Shared Buffer”, “Block Crosspoint Queueing and Crosspoint Queueing (CQ)”. For our architecture (see Figure 2), we use the CIOQ as queuing architecture for internally non-blocking.

In fact, Each Electronic Control Unit (CAN Controller Node) produces a class priority of messages. For example, ECU_0 produces high level of priority and ECU_n (n in our model is equal 3) produce low level of priority.

In fact, Produced CAN messages will be queued in the input queue of the incoming interface (If the input queue is full, the packet is dropped.). Therefore, to respect the CAN protocol philosophy, CAN messages will be broadcasted for all output port through crossbar Switched Fabric [8].

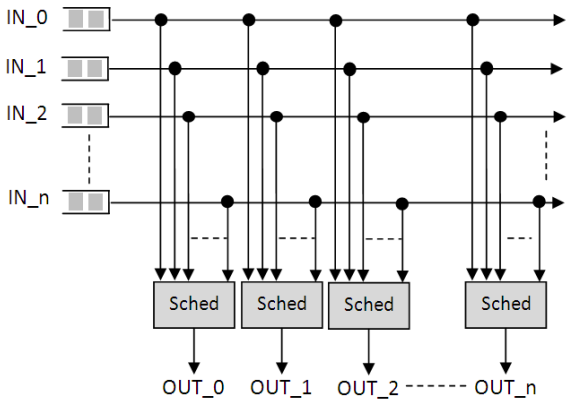

Fig. 1. NxN crossbar Switched Fabric supporting broadcast

Manuscript received and revised February 2012, accepted March 2012

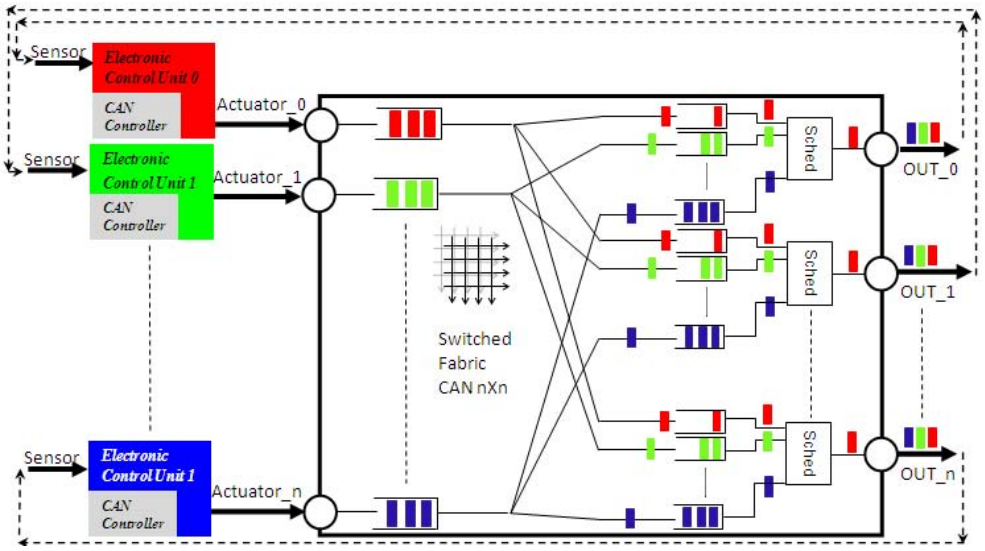

Fig. 2. Switched Fabric CAN Network interconnects

Furthermore, To reduce congested output port and to respect the priority policy, each CAN message will be queued in the suitable output queue of each outgoing interface according to his level priority. (If the output queue is full, the packet is dropped). Then, each output port scheduler will select the message to be sent among the existing CAN message in accordance with his priority. In our work, we modeled the switched fabric CAN Network using stochastic colored Petri. We also evaluated it using CPNTools. Our major contribution is to raise the lack of the bus solutions by proposing switched Fabric CAN architecture and by evaluating its performance. In fact, we demonstrated that CAN based Networks using crossbar Switched fabric [9] have yet a well period before its replacement and it can compete the new sophisticated buses.

Our paper is organized as follow:

- The section 2 gives a short overview of colored Petri net and details the SCPN models of the most important Switched fabric CAN network modules.
- Based on our modeling of the proposed architecture by stochastic and colored Petri nets (SCPN) [10], we present, in the last section, some experiments which are applied to the model.

Three important parameters were measured the throughput, latency time and the loss probability. This work is used to demonstrate the benefit of the proposed architecture. In the fourth section we give some conclusions of our work.

II. Switched Fabric CAN Network Modeling

II.1. Short Recall of Coloured Petri Net

Coloured Petri Nets have been developed by K. Jensen in course of his PhD thesis (Jensen, 1980) to expand the modeling possibilities of classical Petri Nets. Like other forms of Petri Nets a CPN consists of places, tokens, transitions and arcs.

The primary feature unique to CPNs is the inclusion of evolved data structures into tokens [11], [12], [13]. These data structures are called colorsets and resemble data structures in high level programming languages; they can range from simple data types such as integers to complex structures like structs or unions in C/C++. Similar to programming languages it is possible to define variables associated with these colorsets such as linked list and queue.

Some examples of colourset and variable definitions are shown in Fig. 3. Tokens as well as places of a CPN are always associated with a colourset and a place may only contain tokens of the same colourset as its own. To well understand the SCPN models of our Switched CAN controller, we give a short recall of CPN concepts.

```
▼Color Declarations
  ▼colset bit =with Res|Dom;
  ▼colset byte= list bit with 8..8;
  ▼colset Data= list BYTE with Total_Byte..Total_Byte;
▼Variable Declarations
  ▼var data: Data;
```

Fig. 3. Colored and variable definitions

The places in a CPN are depicted as ellipses (Fig.4) with the name of the place written into it and the associated colourset (Id) below. A token in a CPN is represented by a small circle (Fig. 4). Its value (the data stored in the token) is shown in a rectangle attached to the circle. A number in the circle denotes the number of tokens with the same value. Figure 4 for example shows a place called *Buffer_Node_1* associated with the colourset CAN_Messages and holding one token with a value of:

{{ID=[Dom,Res,Dom,Dom,Res,Dom,Dom,Dom,Res,Res,Dom,Dom,Dom,Res,Dom,Res,Dom,Dom,Dom,Res,Res,Dom,Res,Res,Res,Res,Res,Dom,Res],DATA=[byte(4),b

yte(6),byte(5),byte(1),byte(5),byte(6),byte(6),byte(1)],TS =0}].

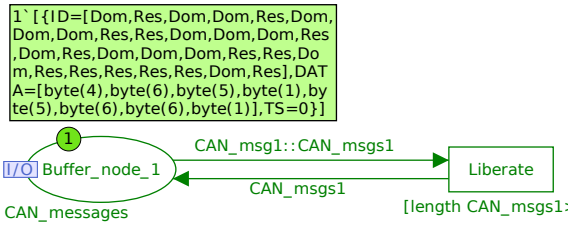

Fig. 4. Graphical representation of a place in CPN

Transitions in a CPN are represented by rectangles (Fig. 5) and can access the data stored in tokens by mapping tokens to variables. There are two possibilities to access this data:

- Guard conditions: The transition is enabled only if a specific condition – called a guard condition – regarding one or more variables is met. Guard conditions are encased in brackets and written above the transition (Fig. 5).
- Transfer function: The transition reads and writes variables according to a specified function that can range from simple addition of values to complex conditional commands.
- Transfer functions consist of the definition of input () variables, output () variables and the commands to be carried out (action ()) and are attached below the transition (Fig. 5).

The example depicted in Figure 5 shows a transition that only fires if the length of variable *CAN_msgs* is less than the value *FIFO_length* and generates an output variable *CAN_msg* without taking any input variables (Fig. 5), the variable *CAN_msg* is filled with the return value of the function defined in the action part, *new_MSG_0*, which in this case is defined in the CPNtools area Declarations.

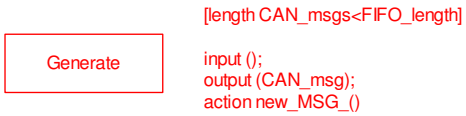

Fig. 5. Transition Generate with guard condition and transfer function

Places and transitions in a CPN are linked by arcs. Arcs in a CPN can be unidirectional or bidirectional. Unidirectional arcs transfer tokens from a place to a transition or vice versa (Fig. 6).

Fig. 6. Unidirectional arc with mapping to value server

Bidirectional arcs transfer the same token from a place to a transition and back (Fig. 7). Arc inscriptions define the mapping of tokens to variables. An inscription can either be a constant value (Fig. 6) or a variable of the colourset that is associated to the place the arc is connected to (Fig. 7). If all places connected to a transition by unidirectional input arcs or by bidirectional arcs hold tokens and its (optional) guard condition is met, the transition is said to be enabled. In case of more than one enabled transition in a CPN the one to fire is chosen randomly. Later on, we will add more places to our controller models to avoid arbitrary transitions.

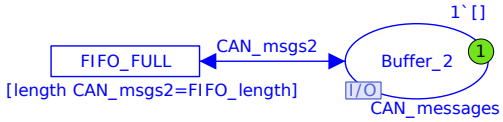

Fig. 7. Bidirectional arc with mapping to variable CAN_msgs

For an analysis of clocked systems it is possible to define timed colourset, defined by the keyword timed (Fig. 8) and transition or arc delays marked by the characters @+ (Fig. 9).

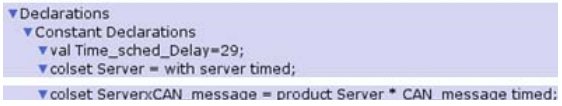

Fig. 8. Timed colourset

If a colourset is defined as timed, a timestamp is added to the tokens of this colourset. The timestamp cannot be accessed by guard conditions or transfer functions. When using timed colourset the firing of transitions depends on a global clock counter. Transitions can only fire if the clock value is the same as the largest timestamp of its input tokens. When a transition fires with a timed arc, the timestamp of its output token is the sum of the current clock value and the arc delay, in the example in Figure 9 this delay is *Time_sched_delay* clock cycles.

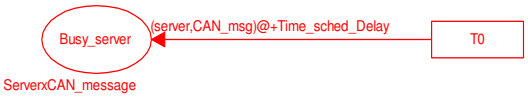

Fig. 9. Timed arc inscription

II.2. SCPN Based Switched CAN Controller Model

In order to facilitate modeling of Switched CAN Controller a modular approach was taken making use of hierarchical CPN models. The model of the Switched Controller is built following a hierarchical and modular architecture.

The root of the hierarchical representation of the model is shown in the Figure 10.

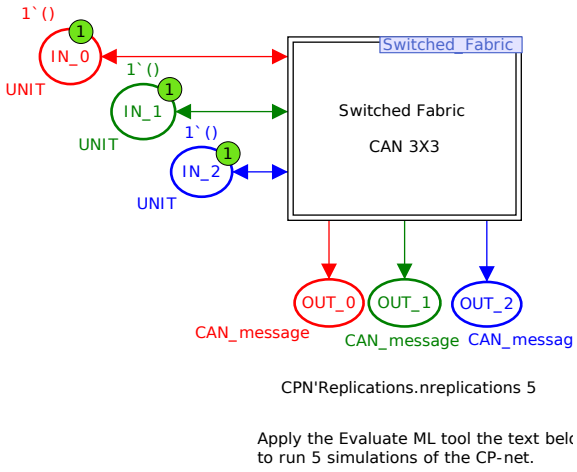

Fig. 10. Root level of the model

The Switched Fabric CAN 3x3 whose activity is modeled by the transition *Switched_Fabric* transmits the CAN message via the Switch Fabric. The places *IN_i* and *OUT_i* (i can be a value between 0:2) play the role of inputs/outputs for sub-models.

Nodes in CAN are identified by their identifier (in this model, colourset *Id* is a list of 29 bits). The colousets and variables used in this model are shown in Figure 11.

Messages sent through the Switched Fabric CAN are represented by tokens of the colourset *CAN_message*. This colourset is a record of the colourset *Id* that designates the message priority and the colourset Data which represent the data field to be transmitted and the colourset *TS* for saving the time stamp for the birth of the message.

Fig. 11. Colousets for CAN Network model

The variables (*CAN_msg*, *CAN_msg1* and *CAN_msg2*) are of type of the colorset *CAN_message*. This variable models the messages which cross the different sub-models of Figure 12 (*Node_i*, *Broadcast_i*, *FiFo_i_j* and *Scheduler_i*). The Switched CAN network model in Figure 12 is composed of three nodes. Each node is represented by a transition and two places. The transition called *Node_i* (i can be a value between0:2) is a hierarchical transition which describes the messages generation within the node, how the messages are stored in buffer. The place *Buffer_Node_i* is used to store the messages already generated. This place is configured with colourset *CAN_messages* which is a list of colourset *CAN_message*. When a token is present on this place (*Length CAN_msgs* >0) a message is ready for sending. This last fires the hierarchical transition *Broadcast_i*. The originated message is duplicated in three places, one for each output port of the Switch fabric. According to the priority which is associated to the messages (defined by their ID), the messages are stored in the FIFO queue (there is as many queue as of priorities). In this model, three levels of priority are defined: 0: high level of priority; 1: medium priority and 2: low priority.

FiFo_i_j is a substitution transition which presents the queue of *input_i* for the *output_j*. Finally, the scheduler processes the different messages according to its scheduling policy.

II.2.1. Generating CAN Messages (Node_i Transition)

As shown in Figure 13, the load source is modeled by the place *Next* and the transition *Generate*. Initially the place *Next* contain one token and is connected via two arcs to the transition *Generate*, the arc from *Generate* to *Next* is timed using the exponential random function. Thuswe can have a random message with parameterized inter arrival period using the value *lambda_i* (*lambda_0* for node 0: Figure 13). The place *Buffer_node_i* is used as messages buffer, sized of 4 in our case, to decouple the source message from the Switched CAN controller. When the load source increases and no room is available in the buffer, an overflow occurs and the transition *FIFO_FULL* fires leading to lose the last generated message (due to a congestion or excessive load).

II.2.2. Broadcasting of CAN Message (Broadcast_i Transition)

The set of broadcasting message is represented by the model described in the Figure 14. Transition *Liberate* models a message coming from *Buffer_node_i*. The *Buffer_node_i* place is a list of CAN_Message. When the list length is not null (i.e there is at least a message to send), the *liberate* Transition can be fired if the line is free (there is a token in *Line_Free* place). Otherwise, message coming from *Node_i* has to be delayed *Transfer_Data_Delay* until the previous message will be liberated. If the message is liberated, the *Server* token will be moved from the place *Line_Free* to *Line_Busy*.

Then the messages will be duplicated in the right place (queues) according to their priorities. For example, message of medium priority (*CAN_msg1*) will be routed to the places *m1_0*, *m1_1* and *m1_2*. The CAN message *m1_i* will be queued in the queue 1 of the output port i (*OUT_i*).

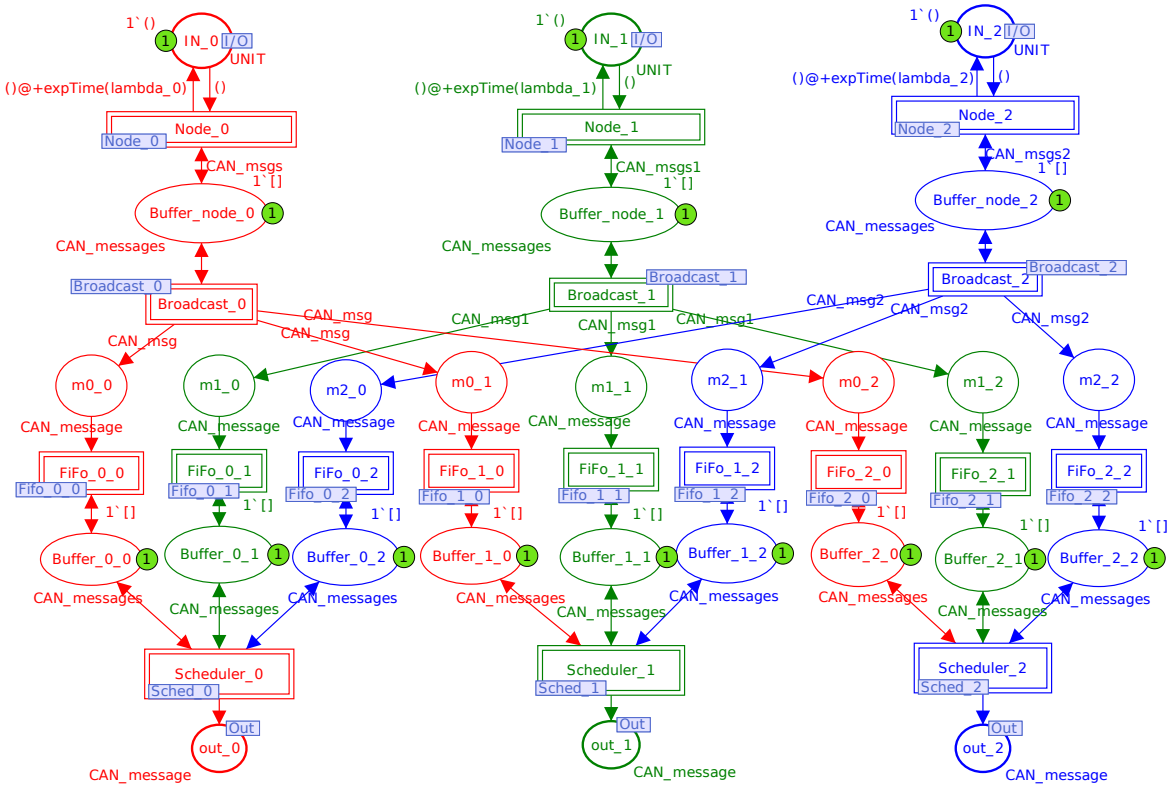

Fig. 12. SCPN Switched CAN Networks model with three nodes having three different priority classes

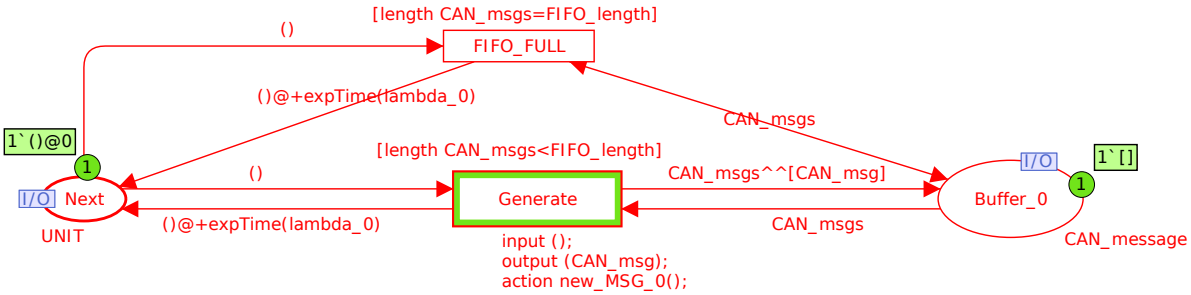

Fig. 13. Generation of CAN Message (Node_0)

II.2.3. Storing CAN Messages (FiFo_i_j)

FIFO model is represented in the Figure 15. It processes the messages in the order of their arrival.

The function of the transition *Arrive* is to concatenate incoming message (*CAN_msgi*) to the *Buffer_i_j*.

Buffer_i_j is a place having *CAN_messages* as colourset: i indicates the level of the message as it was explained previously and j indicates the output port of the model. Thus *Buffer_i_j* is the queue of Message i of the output port j (*OUT_j*).

When the buffer is full (in our model *FiFo_length= 4*), the transition *Fifo_i_j_Full* is fired and the incoming message will be rejected.

II.2.4. Scheduling CAN Messages (Scheduler_i)

The model of the Figure 16 describes the behavior of a static priority scheduling. The type of messages is classified in three groups:

- High priority messages: These messages are generated by Node_0 and are modeled in the place *Buffer_i_0* as a list of *CAN_Message* (*CAN_msgs*) in the Scheduler of the output port i (*OUT_i*).
- Medium priority message: Those are generated by *Node_1* and are modeled in the place *Buffer_i_1* as a list of *CAN_Message* (*CAN_msgs1*) in the *Scheduler_i*.
- Low priority message; those are generated by *Node_2* and are modeled in the place *Buffer_i_2* as a list of *CAN_Message* (*CAN_msgs2*) in the *Scheduler_i*.

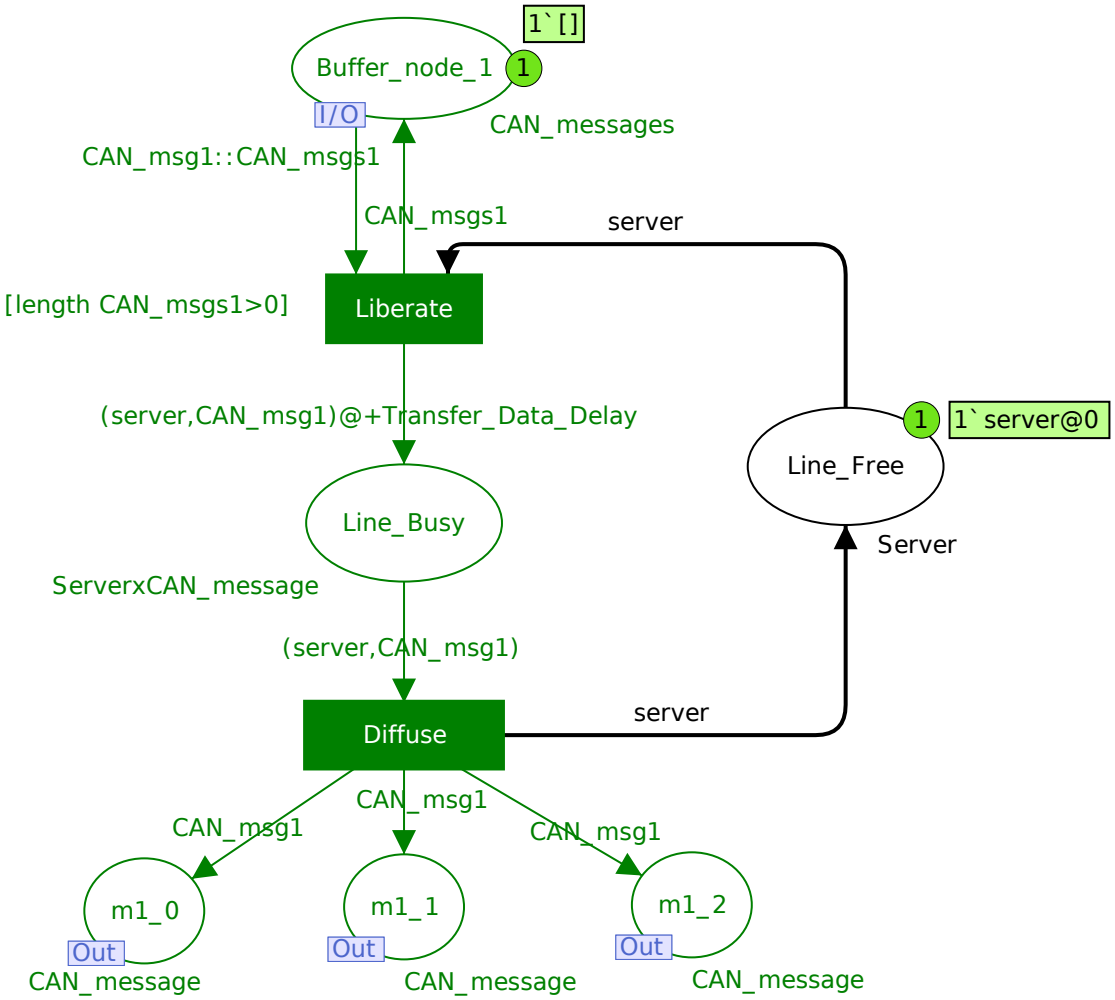

Fig. 14. Broadcasting CAN Messages generated by Node 1

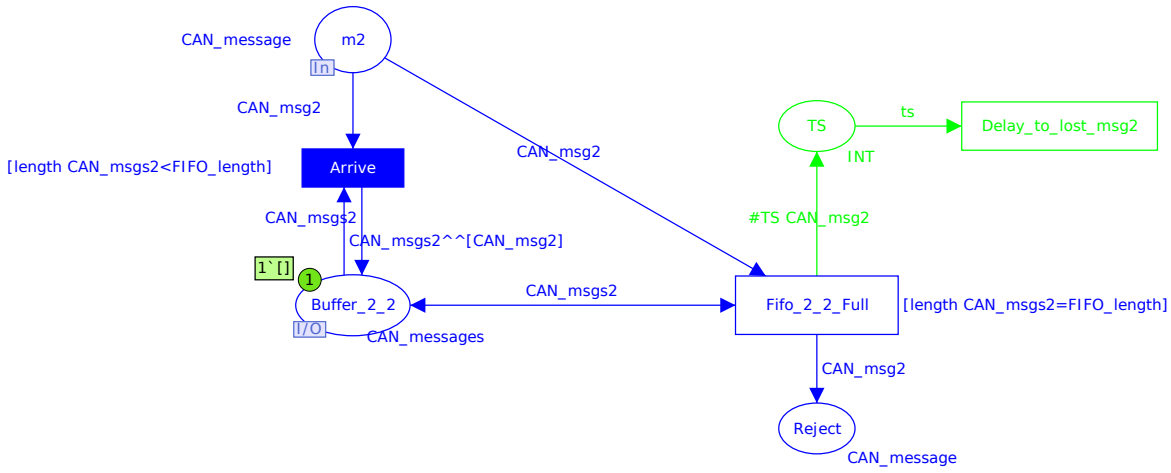

Fig. 15. Storing CAN Messages generated by Node 2 in the queue 2 of output 2

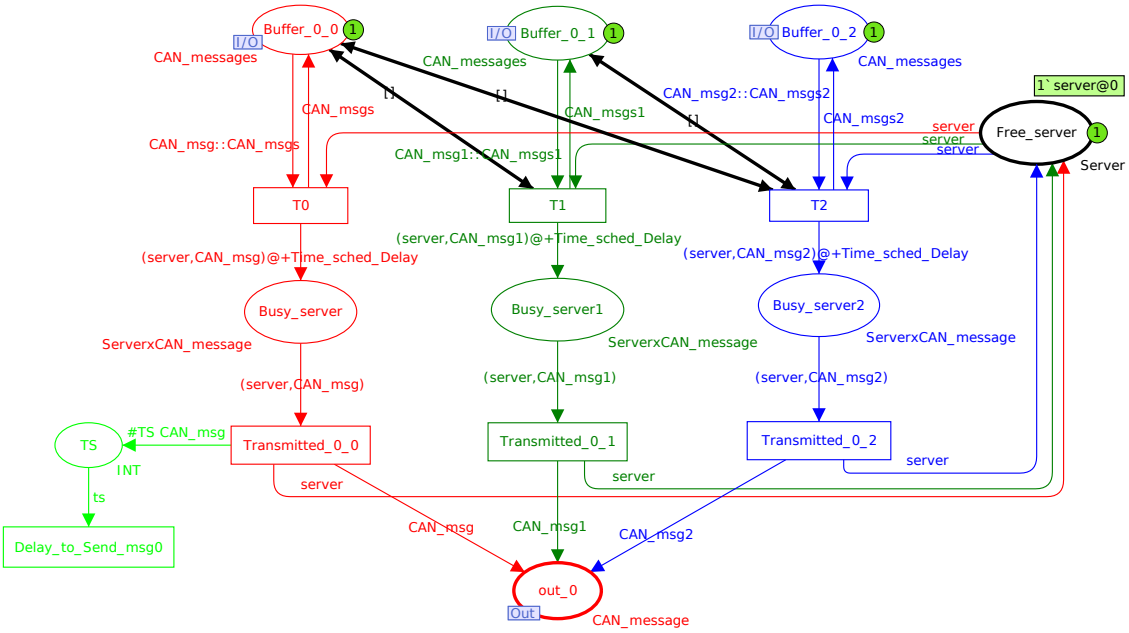

Fig. 16. Scheduling CAN Messages at the Scheduler of Output 0

A message with lowest priorities can be delayed by the other packets due to the non-preemptive characteristics of this kind of message scheduling algorithms [14]. For example, in the case of Scheduler_0, the messages of Buffer_0_2 (low priority messages) has to wait until the messages of Buffer_0_0 and Buffer_0_1 (high and medium priority messages) are fully transmitted. Then, the messages of Buffer_0_1 (medium priority messages) has to wait until the messages of Buffer_0_0 (high priority messages) are fully transmitted. This scheduling policy is modeled using bidirectional arcs between buffers places and the transitions *T1* and *T2*. These arcs are inhibitor arcs. The method of usage of inhibitor arc is described in more details in [13]. When there is at least a message to transmit, the Ti transition can be fired if the server is free what it means that there is a token in *Free_server* place. Otherwise, the following message (according to the algorithm described above) have to be delayed *Time_Sched_Delay* until the previous message is fully transmitted. After the transmission of the message, the Server token will be moved from the place *Busy_sereri* to *Free_server*. The interest of the static priority algorithms is that it is easy to implement. Other Scheduling algorithms can be studied in future works.

III. SCPN Simulation and Evaluation of Crossbar Switched Fabric CAN Network

The objective of this part is to simulate Switched Fabric CAN and to evaluate it. For analyzing different traffic experiments, the nodes in the network are configurable by:

- The duration of Data bursts (measured in clock cycles) to be transmitted is equal to Transfer_Data_Delay which is equal to *Total_byte* x 8= 64μs.
- The duration of scheduling to choose which CAN Message have to be transmitted between all the CAN messages is equal to *Time_Sched_Delay* which is equal to the length of the Id(29 bit) x1µs =29 µs.
- The average delay between two successive request for sending message (*lambda_i* for each node i)
- Each node represent a class of message object: node 0 is the class of high priority messages enoted by two dominant bits *Dom,Dom* on the beginning of its ID. The second node 1 is the class for medium priority messages strating with *Res,Dom* or *Dom,Res* on its ID field. The last node 2 is for low priority messages denoted by *Res,Res* value on the beginning of his ID field.

Performance measures obtained for the SCPN model are the number of requests sent, throughput, latency and the loss probability. The number of requests sent is measured by the addition of counting the times the transitions *Generate* and *FIFO_FULL* fire (see Figure 13 Node 0). The throughput is the total messages sent divided by the total messages generated in the time spent [15]. Latency in this case corresponds to the average time spent from the birth time of message (saved in the *TS* variable) until its full transmission.

Loss probability is the total message lost (due to full FIFO in input buffer or output buffer) divided by the total message requested. In CPNtools there is a monitor tools which provide statistics measurement on either places or transitions [12].

A group of monitor called *NB_MSG_i_G* and *NB_MSG_i_Not_load* (i is in 0..2) are used to measure the occurrence transitions of *Generate* and *FIFO_FULL* for each node. The total number of those occurrences represents the total message generated by the application layer. Another group monitor denoted by *Delay_to_S_msg* is used for calculating the average delay that a message takes from generation at transition *Generate* to the transition *Transmitted_i_i*. This group is a data collector monitor and the observer function ensures the calculation of the time that a message spends using the information on the TS place (see Figure 15).

The average of this monitor is in fact the latency delay. A group monitor named *NB_MSG_LOST* counts the occurrence transitions of *FiFo_Full_sched_i* for node1 and node2.

In context of network performance analysis experiments have been conducted with the model.

The experiments are defined as follow: The total inter-arrival time for all nodes is increased from 10 packets/s to 100000 Packets/s. Load varies in all nodes with the same proportion by the following manner:

- $\lambda 0 = \lambda r0$
- $\lambda 1 = \lambda r1$
- $\lambda 2 = \lambda r2$

r0, r1and r2 are chosen to ensure that the higher priority always guaranties the lowest latency. It is well known that the condition $r0 \leq r1 \leq r2$ must be respected. In our experiment we chose r0=0.2, r1=0.3 and r2=0.5. The experiment treats the influence of low priority nodes to highest nodes and focuses on the impact of high priority nodes to the low priority nodes.

CPNtools does not support non terminating simulation to investigate the steady state of the modeled system. However a group monitor is used to detect network stability. In fact such condition uses complicated algorithm not yet supported by the tool. Using the available monitor features of *CPNtools* we create conditions to know when the network is stable. The basic idea was to compare the last latency delay to its average when is too close to it we increment a counter. If the counter reaches a threshold we stop the simulation. The stop condition must be checked for all the nodes of the network. The group monitor *Steady_Network* ensures the stop condition of the simulation. It consists of seven monitors and is as follow: two for each node, one for calculating the difference of the last latency delay to the average; the second, *msg_i_c,* checks if the difference is less than the threshold (chosen 10 µ seconds). When the condition is true this monitor returns one. If not returns zero. The monitor *Stop_steady_Network* is a breakpoint monitor which checks if for each node we collect a satisfied count number (chosen 1000) of the *msg_i_c*. To collect statistically reliable date, we run 5 simulation replications using the function *CPN'Replications.nreplications 5* (see Figure 10).

III.1. Throughput

Figure 17 shows the throughput of the nodes versus the total requested load. We can see that the node 0's throughput is constant (100%) until 10000 Packets/s and is slightly affected at all values of the requested load. In fact, node 0 is a high priority node so its message will always win the Scheduler priority even if the throughput increases.

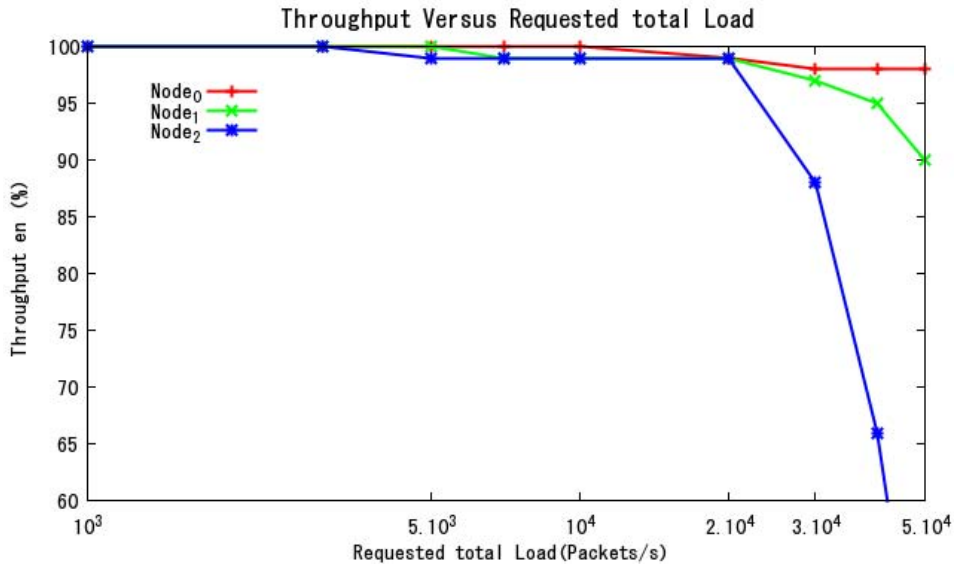

Fig. 17. Throughput of all nodes versus the total load

However the others nodes can't achieve the requested throughput due to an increase of high priority messages.

The node 1's throughput is affected by the presence of higher messages priority from the load of 7000 Packets/s.

The influence is more important for the lowest priority messages (node 2). We note that at 30000 packet/s, the throughput of node 1 and 2 is affected and achieve 97.3 and 88.5 % respectively. Then at 50000 packet/s, we see that the throughput of node 1 and node 2 is more affected and achieve only 90.6 and 38.5 % respectively.

III.2. Latency Delay

Concerning Latency Delay, 93 µs is the minimum time delay needed to transfer a packet over the crossbar Switched Fabric.

The Figure 18 shows that the latency (for all the nodes) starts to be constant, 93 µs, from 10 packets/s till 3000 packets/s. From the load 5000 packets/s, the latency delay increases more for nodes 1 and 2 than node 0 because of the high priority of its message.

For example, at 10000 packets/s, the latency of node 0, node 1 and node 2 reaches 100.83, 103.79 and 110.54 µs respectively. But at 20000 packets/s, thelatencyof node 2 considerably increases comparing to the others one.

III.3. Loss Probability

Figure 19 presents the global loss probability on crossbar Switched Fabric CAN network and the different message classes. We can see that the loss probability increases when the node has lowest priority. In fact, Node 2 and Node 1 lost easily its messages with the presence of higher priority messages produced by Node 0.

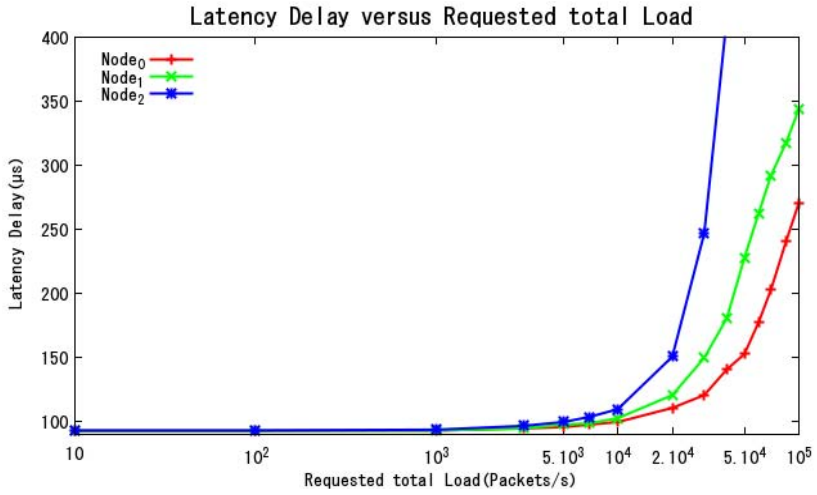

Fig. 18. Latency delay for all nodes versus the total load

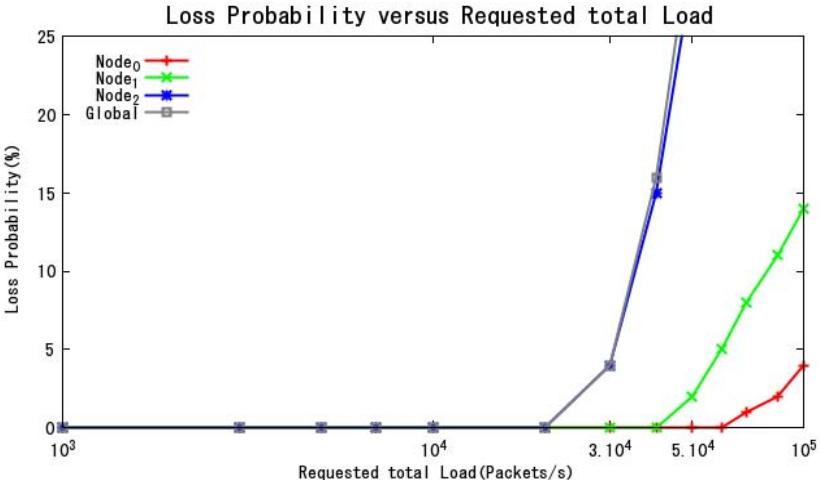

Fig. 19. Loss probability of all nodes versus the total load

The Figure above shows that the global loss probability increases from the load of 30000 packets/s. In fact, it reaches 16.2 % at 40000 packets/s distributed between medium and low priority nodes: 0.9 % for Node 1 and 15.3 % for Node 2.

Up to this result, we note that the crossbar switched fabric is more reliable than the bus for the CAN protocol.

IV. Conclusion

In this paper, Switched Fabric CAN architecture is presented and modeled by CPNTools. The SCPN model of the Switched Fabric Controller is presented with three nodes at transmitter side using a switch fabric (3x3). For that, three message priority classes were treated with a clear representation on ID field. The model focuses on queueing, broadcasting and scheduling mechanisms which are the keys factor for the proposed architecture.

The evaluation of throughput, latency and loss probability obtained after simulations reflects the real time aspect of the proposed architecture, its reliability and its high throughput values. The benefit of this work is to demonstrate that CAN with a crossbar switched fabric has yet a well period before its replacement and CAN controller manufactures do not afraid for its production. A comparison with Bussed CAN controller [1] will be studied in the future works.

References

[1] Oussama KALLEL, Sofiene DRIDI, Salem Hasnaoui "Modeling and Evaluating a CAN Controller Components Using Stochastic and Colored Petri Nets", *International Review on Computers and Software (IRECOS),* Vol.4 N.1, pp 142-151; January 2009.

[2] Rojdi Rekik, Tarek Guesmi, Salem Hasnaoui "Challenges in the Implementation, the Configuration and the Evaluation of a QoS-enabled Middleware for Real-Time Embedded Systems", *International Review on Computers and Software (IRECOS),* vol. 3, n°3, May 2008.

[3] Brett Murphy, Emmanuel Eriksson. "Fabrics and Publish-Subscribe Schemes: A Net-Centric Blend" *COTS Journal Oct. 2009 http://www.cotsjournalonline.com/articles/print_article/100148*

[4] Salem Hasnaoui, Oussema Kallel "A proof-of-concept Implementation of Modified CAN Protocol on CAN Fieldbus Controller Component"; Accepted oct. 2004, Revised Jan. 2005*; AMI. Journal, Ref 03/08.*

[5] Marko Bago, Siniša Marijan, Nedjeljko Perić; Modeling Controller Area Network Communication*, Proceedings of the Ninth Workshop on Practical Use of Coloured Petri Nets and the CPN Tools,* October 20-22, 2008.

[6] CUI Lian-cheng, ZHAO Zheng-fang, XU Xiao-ju2, WU Fang-ming, SHAN Wei-zhen "Real Time Performance Analysis of CAN Bus Based on TimeNET" *The 3rd International Conference on Innovative Computing Information and Control (ICICIC'08),* 2008.

[7] Prodanov, W.; Valle, M.; Buzas, R.; Pierscinski, H."A Mixed-Mode behavioral model of a Controller-Area-Network bus transceiver: a case study" *Behavioral Modeling and Simulation Workshop, 2007. BMAS 2007. IEEE International* Volume, Issue , 20-21 Sept. 2007 Page(s):67–72

[8] Arshad, Nauman, Stewart Dewar, and Ian Stalker. "Serial Switched Fabrics Enable New Military System Architectures." *COTS Journal Dec. 2005 <www.cotsjournalonline.com/home/article.php?id=10043>*

[9] Dr. Rajive Joshi," Using Switched Fabrics and Data Distribution Service to Develop High Performance Distributed Data-Critical Systems" *The Journal of Defense Software Engineering.* April 2007.

[10] K. Jensen , "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*" (Monographs in Theoretical Computer Science, Springer-Verlag, 1997).

[11] Ehrig, H.; Juhas, G.; Padberg, J.; Rozenberg, G. (Eds.), "*Unifying Petri Nets - Advances in Petri Nets*" (Springer-Verlag, 2001).

[12] A.V. Ratzer, L.Wells, H.M.Larsen, M.Laursen, J.F.Qvortrup, M.S.Stissing, M. Westergaard, S. Christensen, K.Jensen. CPN Tools for editing, simulating, and analysing Coloured Petri Net. *LNC, 2679:450-462, 2003.*

[13] Design/CPN Manuals. Meta Software Corporation and Department of Computer Science, University of Aarhus,Denmark. *On-line version: http://www. daimi.aau.dk/designCPN/.*

[14] John D. Pape, "Implementation of an On-chip Interconnect Using the i-SLIP Scheduling Algorithm". *(The University of Texas at Austin Chap 3 pp 11-15: December 2006)*

[15] Salem Hasnaoui, M. Hamdi, S. Tahar, "Throughput Performance Analysis of Wireless Control Area Networks and its Coupling to the Latency Time", *accepted for publication as a regular paper in Wireless Communications & Mobile Computing (WCMC) John Wiley Interscience Journal, April 2006, Revised September 2006, Ref WCMC-1105-353.To appear in 2009 (Editor's letter).*

Authors' information

Mohamed Mazouzi was born in 1980 in Sfax, Tunisia. He received the degree in computer science engineering from The National School of Computer Sciences, Tunisia, in 2004, and the postgraduate research degree in New Technologies of Dedicated Computer Systems, in 2006, from National School of Engineering of Sfax. Since 2006, he is a PHD student in the Computer and Embedded Systems laboratory of National School of Engineering of Sfax. Since 2006, he occupied the post of a Technologist at the Higher Institute of Technological Studies of Sfax in Computer Sciences and High Technology Department. His research interests are industrial networks for real-time and distributed applications.

Oussama Kallel was born in 1977 in Sfax, Tunisia. He received the degree in electrical engineering from National School of Engineering of Tunis, Tunisia, in 2000, and the postgraduate research degree in communication networks, in 2002, from National School of Engineering of Tunis. In 2009, he obtained his PhD degree in telecommunications at National School of Engineering of Tunis (El Manar University of TUNISIA). Since 2003, he occupied the post of an assistant at the Science Faculty of Bizerte in Physic department (Electronics Unit). His research interests are industrial networks for real-time and distributed applications and their implementations on FPGA component.

Salem Hasnaoui is a professor in the Department of Computer and Communication Technologies at the National School of Engineering of Tunis. He received the Engineer diploma degree in electrical and computer engineering from National School of Engineering of Tunis. He obtained a M.Sc. and third cycle doctorate in electrical engineering, in 1988 and 1993 respectively. The later is extended to a PhD. degree in telecommunications with a specialization in networks and real-time systems, in 2000. Salem is author and co-author of more than 120 refereed publications, more than 30 papers in international journals, a patent and a book. His current research interests include real-time systems, sensor networks, network controllers and multicores, QoS

control & networking, adaptive distributed real-time middleware and protocols that provide performance-assured services in unpredictable environments. Prof. Hasnaoui is the responsible of the research group "Networking and Distributed computing" within the Communications Systems Laboratory at the National School of Engineering of Tunis. He served on many conference committees and journals reviewing processes.

Mohamed Abid is currently professor at Sfax University in Tunisia. He holds a Diploma in electrical engineering in 1986 from the University of Sfax in Tunisia and received his PhD degree in computer engineering in 1989 at University of Toulouse in France. His current research interests include Hardware-Software System on Chip co-design, reconfigurable FPGA, real time system and embedded system. Dr. Abid has authored/co-authored over 150 papers in international journals and conferences. He served on several technical program committees and as a co-organizer of several international conferences. He also served Guest Editor of special issues of two international journals.