Formal Specification and Verification of a Delta-MIN Based Interconnection Architecture for MPSoC

Maïssa Elleuch, Yassine Aydi, Mohamed Abid
CES-National Engineering School of Sfax, Tunisia
maissa.elleuch@gmail.com, yassine.aydi@oous.rnu.tn, mohamed.abid@enis.rnu.tn

Abstract

The design of multiprocessor system-on-chip has performance constraints which must be satisfied by the communication architecture. Multistage interconnection networks have been frequently proposed as connection means in classical multiprocessor systems. They are generally accepted concepts in the semiconductor industry for solving the problems related to on-chip communications. This paper proposes a methodology for the extension of a generic model (GeNoC) describing on-chip communications. At the generic level, the topology component and an extended routing function are defined and implemented in the ACL2 theorem proving environment. We achieve the validation of the extended model on a Delta multistage interconnection networks case study. We thus show the utility of the approach to give a more realistic model describing the communication architectures.

1. Introduction

The future generation of platforms on multiprocessor systems-on-chip (MPSOCs) must satisfy many critical requirements: they have to be energy efficient, cheap, reliable, and must offer sufficient computing power for advanced and complex applications. To satisfy all these constraints simultaneously, future MPSOCs must integrate several types of processors and data memory units, adding more flexibility and programmability to these devices [1]. Therefore, researches were focused mainly in squeezing computing and controlling power on a system on chip (SoC). As a result, many MPSOC platforms have emerged [2].

A key step in the design of such systems is the choice of the communication architecture. Indeed, this communication architecture must support the entire inter-component data traffic and has a significant impact on the overall system performance [3]. As a promising alternative, Networks on Chip (NoCs) have been proposed by academia and industry to handle communication needs for the future multiprocessor systems-on-chip [4]. In comparison with previous communication platforms (e.g., a single shared bus, a hierarchy of buses, dedicated point-to-point wires), NoCs provide enhanced performance and scalability. All NoCs advantages are achieved thanks to efficient sharing of wires and a high level of parallelism [5].

Multistage Interconnection Networks (MINs) have been used in classical multiprocessor systems. As an example, MINs are frequently used to connect the nodes of IBMSP [6] and CRAY Y-MP series [7]. Further on, MINs are applied for networks on chip to connect processors to memory modules in MPSOCs [8]. These architectures provide a maximum bandwidth to components (processors, DSP, IP...), and minimum delay access to memory modules. A MIN is defined by, its topology, switching strategy, routing algorithm, scheduling mechanism, fault tolerance [9], and dynamic reconfigurability [10].

Another basic step in the design of an MPSoC is the verification of the whole system, and especially of the selected communication architecture. Traditionally, this verification is synonym with simulation [11] which consists on the performance evaluation of the system [12]. However, such technique provides partial verification, so it cannot cover all design errors or detect undesirable situations (deadlock, starvation). The new trend is then to adopt formal verification, which is based on using methods of mathematical proof to ensure the quality of the design, improve the robustness of the system, and speed up the development [13].

In general, two methods are applied in formal verification: model checking, and theorem proving. The first one consists in an exploration of the all system states to check the availability of a specified property. The second performs the proofs of theorems about a given model of the system. Although model checking is automatic and fast, it has many drawbacks like the problem of states explosion due to the use of finite state machines. Therefore, theorem proving being independent of the system size, is much more efficient in particular when applied at a high level abstraction.

A Formal specification and verification of Delta MINs for MPSOC in the ACL2 logic is investigated in this paper. Section 2 discusses related work. In section 3, the multistage interconnection networks architecture is introduced. Next, a formal approach to specify on-chip communications is detailed. Finally, we describe how to apply this formalism in a Delta multistage interconnection networks case study.

2. Related work

Various works have been recently done to formally verify on-chip communication architectures. They can be classified into two categories: specific and generic.

2.1. Specific formalization

Roychoudhury and al. verify the protocol AMBA AHB [14]. The main contribution of this work is to have formally validated a protocol involving pipeline in operations. By using the SMV model checker, a scenario of starvation was detected. The same bus has been also verified by Amjad [15]. A simplified model of the AMBA bus has been implemented and validated by using model checking and theorem proving. Gebremichael and al. [16] verify the absence of deadlock in the Æthereal protocol of Philips by developing a formal abstract model via the PVS model checker.

All the works described above develop a dedicated formal model of the system. In addition, the model is generally described at the RTL level which is a low abstraction level. Therefore, given the increasing complexity of future communication architectures for MPSOC, it becomes complicated to achieve such formal verification especially through model checking.

2.2. Generic formalization

The generic formalization is based on a Generic Networks on Chip model denoted GeNoC [17]. GeNoC takes into account the common components of any on-chip interconnection architecture, and models them in a functional style through four functions: "Send", "Recv", "Routing" and "Scheduling". The two first functions describe the interfaces of any network, while "Routing" and "Scheduling" embody respectively the routing algorithm and the switching technique. Each of these functions doesn't have an explicit definition but is constrained by properties modelled by theorems, called also proofs obligation.

The main GeNoC function is illustrated in figure 1. It takes as arguments the list of requested communications (messages to send) and the characteristics of the network, and produces as result two lists: successful communications and aborted ones. The correctness of this function is guarantee through a key theorem expressing that a sent message is always received by its right destination without any modification of its content. Verifying any instance of a given NoC is possible through the GeNoC approach, provided that the NoC components meet the generic constraints.

The GeNoC model has been implemented in the ACL2 theorem proving environment. ACL2 (A Computational Logic for Applicative Common LISP) is a tool involving a mathematical logic of the first order and a theorem prover [18]. To prove one theorem, ACL2 uses various techniques like rewritten, simplification by the repeated substitution of equals for equals, the decision-making procedures, and mathematic induction.

By applying the GeNoC approach, several NOCs (Octagon, Mesh 2D) were specified and validated [17]. The GeNoC model also, has been extended to verify the Hermes NoC [19].

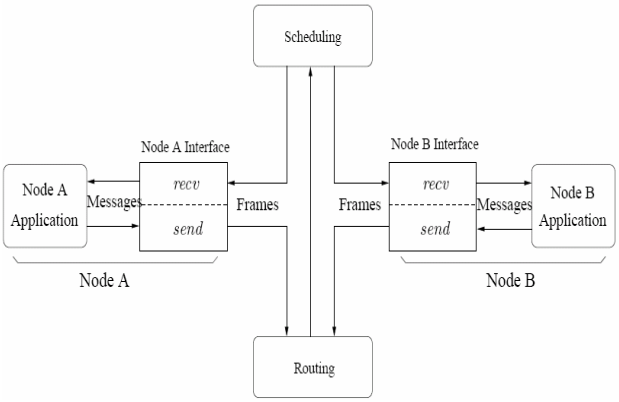

Figure 1. GeNoC: a generic network on chip model

2.3. Discussion

Compared to previous formal works in the context of on-chip communications, GeNoC is innovative. Indeed, it has the specificity to be generic and it doesn't make any assumption on topology, routing algorithm and scheduling policy of the NoC.

In order to deal with our work in a generic approach, we choose to apply the GeNoC methodology to specify on-chip communications based on Delta-MIN. As defined in GeNoC, the routing function takes as input only the set of nodes (*NodeSet*). This function supposes the existence of connections between two successive nodes of a computed route. In this case study, routing is done by applying the self routing. Such routing algorithm depends on the destination address. It gives in each stage the port through which the message must be switched. However, it doesn't give any indication about the position of the next switch. Therefore, applying the

actual GeNoC model to validate the Delta MINs is impossible without considering their topology.

The main idea here is to extend the GeNoC model by involving the connection component, so that we describe a more realistic model of on-chip communications. The extension is possible by defining formally and at a generic level, abstract interconnection functions. After that, the networks case study can be validated by applying the GeNoC model extended.

3. MIN Architecture

In this section, we present an overview of the networks used for the specification.

3.1. MIN Components

The common multistage interconnection networks (MINs) used, have *N* inputs and *N* outputs nodes and are built using $r \times r$ switches. Such MINs have N/r switches at each stage, and $log_r N$ stages of switches denoted *d*. The interconnection stages C_i ($0 \leq i \leq d$) are associated by links generated by applying permutation functions. Figure 2 represents a generic model of MINs of size $N \times N$ using crossbars with *r* equals 2.

In a MIN, a path between a source and a target is obtained by operating a switch at stage Stg_i through the upper output if the i^{th} bit of the destination address is equal to 0, otherwise through the lower output.

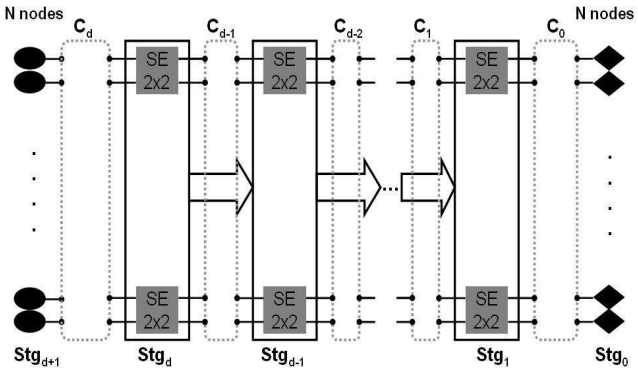

Figure 2. A generic model of MINs

3.1. MINs with Banyan property

We propose in figure 3 a topological classification of MINs. A banyan MIN is a multistage interconnection network characterized by one and only one path between each source and destination. A banyan MIN of size $N \times N$ consists of $r \times r$ crossbars.

An interesting subclass of Banyan MINs is composed of Delta networks. Let denote by: o_i the i^{th} output of a crossbar in a MIN, and by C_j, a crossbar belonging to the stage *j*. So, the Delta property can be defined as follows: if an input of C_j is connected to the output o_i of C_{j-1}, then all other inputs of C_j must be connected to the stage *(j-1)* on outputs with the same index *i*.

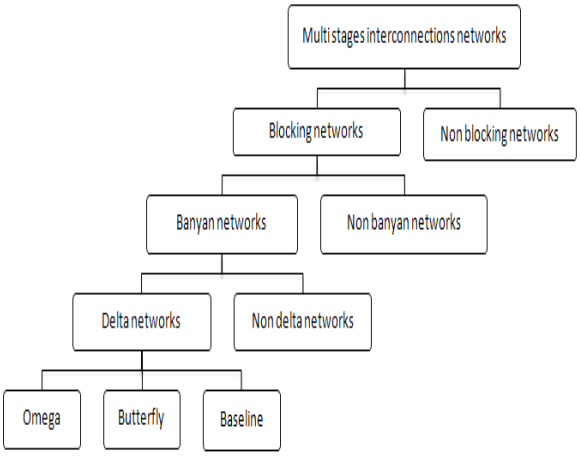

Figure 3. Classification of MINs

3.2. Delta networks

The difference between each of the existing MINs is the topology of interconnection links between the crossbar stages. A study of equivalence of a variety of Delta MINs has been detailed in [20].

We show in table 1 the permutation links of the most popular Delta MIN: omega, baseline and butterfly. In a multistage interconnection network using 2x2 crossbar elements, the common permutation links used are:

- The perfect shuffle denoted σ is a bit-shuffling permutation where: $\sigma^k (x_{n-1}\, x_{n-2} \ldots x_1\, x_0) = x_{n-2} \ldots x_1\, x_0\, x_{n-1}$.
- The butterfly permutation denoted β is a bit-shuffling permutation where:

$\beta_i^k (x_{n-1} \ldots x_{i+1}\, x_i\, x_{i-1} \ldots x_1\, x_0) = x_{n-1} \ldots x_{i+1}\, x_0\, x_{i-1} \ldots x_1\, x_i$.

- The baseline permutation denoted δ is a bit-shuffling permutation where:

$\delta_i^k (x_{n-1} \ldots x_{i+1}\, x_i\, x_{i-1} \ldots x_1\, x_0) = x_{n-1} \ldots x_{i+1}\, x_0\, x_i\, x_{i-1} \ldots x_1$

- The identity permutation denoted I is a bit-shuffling permutation where: $I(x_{n-1}\, x_{n-2} \ldots x_1\, x_0) = x_{n-1}\, x_{n-2} \ldots x_1\, x_0$.

Table 1. Permutation links in Delta MINs

Links permutation	*stage (d+1)*	*stage k ∈ [1..d]*	*stage 0*
Omega	σ^k	σ^k	I
Baseline	I	δ_i^k	I
Butterfly	σ^k	β_i^k	I

4. A generic formalization of communication architecture

We describe below the methodology adopted to specify in formal notations the interconnection networks. We detail a generic topology and extended routing components as extension of the generic model GeNoC.

4.1. The topology component

Taking into account the common components of all networks topologies, we detail in this subsection the formalization of a generic network topology composed of nodes set and connections. As the nodes set was already defined in GeNoC, we focalize on the generalization of connections. We keep the same GeNoC notation of functions and predicates defined for the nodes.

The arrangement of the elements (nodes and links) of a network, especially the physical (real) and logical (virtual) interconnections between nodes, defines its topology. In general, the study of the network physical topology is assimilated to the study of a graph, which vertices are the nodes of the network and its edges are the links connecting pairs of vertices. Traditionally, a graph has been always defined statically by its collection of vertices *(V)* and its collection of edges *(E)* [21]. In contrast, the approach is based on adopting a direct graph for the topology, and identifying the interconnection functions list.

In a direct graph, the edges are oriented from one vertex to another. A vertex *x* of the graph can be connected to one or more other vertices. In general, to generate one edge or link from the vertex *x*, we have to apply a mathematical function designated by *fp*. Such function expresses the relation between the vertex *x* and one of its outgoing edge. All outgoing edges from *x* are the result of the application of a list of functions denoted *lfpx*. For the validity of the generic topology model, we suppose that for each vertex, such mathematical function list exists. The main constraint to check on this graph topology is that a vertex *v* produced by a given function connection *fp*, is really in the nodes set *(NodeSet)*. Figure 4 shows a simple topology graph in which nodes are naturals (*NodeSet = (1,2,3))* and its connections are described by an incremental function *fp* = '+1'.

In the ACL2 logic, we define the function denoted *Gen-Cnx* which generates all the edges of a given vertex *x*. It takes as arguments the vertex and the corresponding function list *lfpx*. The predicate *Validlfp* recognizes a valid list *lfpx*. The first constraint on topology (theorem 1) expresses the correction of the function *lfpx*. Such a function is valid if, for every connection *cnx* generated from a node *x (in NodeSet)* and its corresponding list *lfpx*, the second extremity of *cnx* belongs to the *NodeSet*. The access to the second extremity of any connection *cnx* is possible through the function *ext2*.

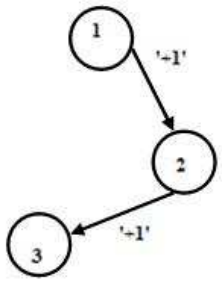

Figure 4. A topology graph

Finally, we define in ACL2 the function nominated *Gen-Top* which generates all the graph edges or links. It takes as parameters the set of vertices *(NodeSet)*, the list of connections functions *(Listfp)* and parameters *pms-top*. The correction of *Gen-Top* is defined by the second theorem (theorem 2). It says that, for all valid parameters *pms-top* (recognized by *ValidParams-top*) and all valid *listfp* (recognized by *ValidListfp*); every connection produced by the function *Gen-Top* is valid, i.e recognized by the predicate *ValidTop*. *ValidCnxp* is the predicate associated to the validity of a connection *cnx*. The definition 1 describes the function *Gen-Top*. *List* is an ACL2 function used to build a list of elements.

Definition 1.

Gen-Top (NodeSet,listfp, pms-top)=

$\wedge$ *(List (Gen-Cnx (x, lfpx)))*

$x \in NodeSet$

$lfpx \in Listfp$

The translation of the generic definitions in ACL2 is possible through the encapsulation principle [22]. This principle introduces under certain constraints, functions symbols without any explicit definitions. The constraints are theorems.

```
(encapsulate (((f x1...xn) => *))
    (local (defun f (x1 ...xn)    β))
    (defthm thm-1 ϕ))
```

When the encapsulated event is admitted, the ACL2 theory is extended by the axiom: *f* is constrained with ϕ. The function *f* doesn't have an explicit definition but we

know that it has the property ϕ. So, a definition of a given function *g* is an instance of the encapsulated function *f*, only and only if, *g* can satisfy the same constraints of *f*. Otherwise, all theorems expressed at the generic level for *f*, have to be proved for *g*.

Theorem 1. Definition of connection functions
$\forall x \in NodeSet, \forall lfpx, Validlfp\ (lfpx)$
$\Rightarrow \forall cnx \in Gen\text{-}Cnx\ (x, lfpx), ext2\ (cnx) \in NodeSet$

In the ACL2 logic:
```
(defthm ext2-lfpx-in-nodeset
 (let* ((nodeset (NodeSetGenerator pms))
        (cnx (Gen-Cnx x lfpx))
        (ext2 (ext2 cnx)))
        (implies (and (ValidParamsp pms)
                      (member-equal x nodes)
                      (Validlfp lfpx))
                 (member-equal ext2 nodeset))))
```

Theorem 2. Definition of connections
$\forall x \in NodeSet, \forall lfpx, Validlfp\ (lfpx)$
$\Rightarrow \forall cnx \in Gen\text{-}Cnx\ (x, lfpx), ValidCnxp\ (cnx)$

In the ACL2 logic:
```
(defthm gen-top-generates-valid-top
 (let ((nodeset (NodesetGenerator pms))
       (top (Gen-Top nodeset Listfp pms-top)))
       (implies (and(ValidParamsp pms)
                    (ValidListfp Listfp)
                    (ValidParamsp-top pms-top))
                (ValidTop top))))
```

4.2. Extended Routing function

As we said earlier, the generic routing function depends only on the set of nodes (*NodeSet*). For each missive (message) denoted *m* from the set of missives *M*, *"Routing"* applies a function ρ that computes all possible routes between the origin of the missive (Org_M) and its destination ($Dest_M$). We have redefined the routing function; so that it takes into account the whole topology (denoted *Top)* which is composed of nodes and connections. In particular, the function ρ must takes into account the *Top*. The new routing function is designated by *Ext-Routing* (definition 2). In this definition, we use Id_M, Frm_M, Org_M, $Dest_M$ which are GeNoC functions giving access to the elements of a missive. A missive is a data type defined also in GeNoC. It represents a message having the form: "*id org frm dest*", where *id* is used to identify the message, *org* is its origin, *dest* denotes its destination and *frm* is the content of the message.

Definition 2.
Ext-Routing (Top,M)=
$\wedge$ *(List* $(Id_M(m), Frm_M(m), \rho(Org_M\ (m), Dest_M\ (m), Top))$
$m \in M$

GeNoC defines three constraints on the *"Routing"* function. The first constraint is related to the validity of computed routes which is expressed by the predicate *ValidRoutep*. It said that any route *r* produced by the function ρ must be valid. So, *r* is valid only if it starts at the source node $(Org_M\ (m))$, terminates at the destination node $(Dest_M\ (m))$, all the nodes of the route *r* are included in *NodeSet*, and *r* includes at least two nodes. We also redefined *ValidRoutep* in order to involve the topology component. The new computed route *r* is no longer just composed of nodes but of connections. We denote the i^{th} element of *r* by *r[i]*, and by *l* the length of *r*. Thus, the new predicate *Ext-ValidRoutep* requires that the first of element of *r* (function *first*) is equal to the origin of the missive $(Org_M\ (m))$, the last of *r* (function *last*) is equal to the destination of the missive $(Dest_M\ (m))$, that *r[i]* is a connection *cnx* included in *Top*, and that *r* has a length *l* greater or equal than 1. *l* is equal to 1 if the origin and the destination are directly connected.

Definition 3. Definition of the route validity
Ext-ValidRoutep (r, m, Top) =

$$\wedge \begin{cases} (First\ (r[0]) = Org_M\ (m) \\ (Last\ (r[l\text{-}1]) = Dest_M\ (m) \\ r \subseteq Top\ \wedge\ (len(r) \geq 1) \end{cases}$$

The constraint concerning the validity of routes is redefined in theorem 3. The GeNoC predicate denoted M_{lstp} recognizes a valid list of missives. We have redefined it to include the topology *Top*.

Theorem 3. Validity of routes produced by ρ
$\forall M, M_{lstp}\ (M, Top)$
$\Rightarrow \forall m \in M, \forall r \in \rho\ (Org_M\ (m), Dest_M\ (m)),$
Ext-ValidRoutep (r, m, Top)

The two other constraints defined on the GeNoC routing function remain almost valid and won't be the subject to modifications.

5. Specification and verification of a Delta-MIN based architecture

In this section, we apply the extended GeNoC model to validate the Delta multistage interconnection networks. We insist on the applying of the generic topology to a Delta MIN case study.

5.1. The Delta MIN topology component

The generic approach requires beginning by identifying the connection functions to apply to generate

all the network connections. Once this list identified, we have to check the three main constraints expressed at the generic level: one constraint defined in GeNoC for the validity of the nodes set; and the two constraints that we have defined for the connections (theorem 1 and 2). The Delta MIN topology as described above (figure 3) is composed of nodes and connections.

-The set of nodes: a pair of coordinates (x y) is used to represent a node in a Delta MIN. The coordinate x is decimal. It represents the stage of nodes to which belongs the node. The Y coordinate is binary and it describes the position of the node within the same stage. The function *gen-nodes-dmin* generates all nodes of the network. It takes as parameters N, the size of the network, and r (r=2), the degree of switches. The validity of these parameters is recognized by the predicate *ValidParamsp-dmin*. We define also another predicate called *dmin-nodesetp* for the whole nodes validity. The nodes set generation is constrained by the theorem 4.

Theorem 4. Nodes set generation

```
(defthm gen-nodes-dmin-correct
 (implies (ValideParams-dmin pms)
          (dmin-nodesetp(gen-nodes-dmin pms))))
```

-Connections: we represent a connection *cnx* in a Delta MIN by a list *((x px) (y py))*, where *x* is the origin of *cnx*, *px* is the port involved in *cnx*, *y* is the second extremity and *py* is the port of *y*. For example, the connection *(((3) (0 1)) L0) ((((2) (1 0)) R0)* denotes that the port *L0* of the switch *((3) (0 1))* is connected to the port *R0* of *((2) (1 0))*. In the case of Delta MIN, the connection functions are always a list of three permutations to apply respectively on the first stage of connection, the middle stages and finally, on the last stage.

In the ACL2 logic, we define the function *gen-cnx-node* that generates all connections of one node *n*. It takes as arguments the node *n* origin of connections, the list of permutation functions, the parameter *d* denoting the stages number of the network, and *r* the degree of the switches. The theorem 5 is an instance of the theorem 1 expressed at the generic level. It checks that every node *ext2* produced by the permutation function σ_{n-1} (modelled by *sigmak*) belongs to the set of nodes (*nodes*). The same constraint must be also verified for the other two permutation functions. We define below the function *gen-top-dmin* (definition 4). It generates all the connections of a Delta MIN by taking as inputs *N* and *r* previously defined, and the type of the Delta MIN. The last parameter is used by *gen-topology* to select the types of permutations corresponding to this network.

Definition 4. Generation of Delta MIN topology

```
(defun gen-top-dmin (nodes lfp d r)
 (if (endp nodes)
     nil
     (let ((n (car nodes)))
(cond

;; connection source-switch
 ((s-node n d)
  (append (list (gen-one-cnx n 'nil (car lfp)))
  (gen-top-dmin (cdr nodes) lfp d r)))

;; connection switch-switch
 ((and (sw-node n d) (> (caar n) 1))
  (append (rev (gen-cnx-node n r (cadr lfp)))
  (gen-top-dmin (cdr nodes) lfp d r)))

;;connection switch-destination
 ((and (sw-node n d) (equal (caar n) 1))
  (append (rev (gen-cnx-node n r (caddr lfp)))
  (gen-top-dmin  (cdr nodes) lfp d r)))))))
```

Theorem 5.

```
(defthm dest-cnx-in-dmin-nodeset
 (let* ((nodes (gen-nodes-dmin pms))
        (cnx (gen-one-cnx ext1 i 'sigmak))
        (ext2 (ext2-cnx cnx)))
 (implies (and  (ValidParamspD pms)
                (member-equal ext1 NodeSet)
                (true-listp i)
                (valid-fp fp))
          (member-equal ext2 nodes))))
```

Finally, we prove the third constraint that expresses the validity of connections (theorem 2 at the generic level), and check the compliance of the definitions with the generic topology component extended.

5.2. The Delta MIN routing component

The routing algorithm used in Delta MINs is the self routing. It depends only on the destination address, called also control sequence. If the corresponding digit of the control sequence is equal to *i*, the message to deliver will be switched to the output *i* of the current crossbar. Here, the routing algorithm must take into account connections. Indeed, the only information of the port through which the message must be switched is not enough. Thus, we must look in the topology for the connection with the current switch as origin.

As defined in ACL2, the routing function *routing-dmin* takes as arguments the list of missives to be routed through the Delta MIN, and the parameters to generate the whole topology. For each missive, *routing-dmin* calls the following function *compute-rte* (definition 5) to compute the route between the origin (*from*) and the destination (*to*).

Definition 5. Function compute-rte.

```
(defun compute-routes-dmin (from to cdrto top)
(if (endp cdrto)
     nil
(let*((bit_rtg (car cdrto))
      (from-a (adapt-node from bit_rtg))
      (next-node (ext2 (rech-top from-a top))))
```

```
(cond

;; destination bit equals 0
((equal bit '0)
 (list* (list from-a next-node)
 (compute-routes-dmin next-node to (cdr cdrto)
top)))

;; destination bit equals 1
((equal bit '1)
 (list* (list from-a next-node)
 (compute-routes-dmin next-node to (cdr cdrto)
top))))))
```

The ACL2 theorem proving environment provides also an execution engine. Thus, we can simulate the execution of the definitions. We present below a simulation of the function *routing-dmin* showing the progression of the list of missives (table 2) through an omega network 8x8, using 2x2 crossbars.

Table 2. The list of missives

id	*origin*	*content*	*destination*
1	((4) (0 0 1))	frm1	((0) (1 0 0))
2	((4) (1 0 1))	frm2	((0) (0 0 1))

The simulation result of this list of missives is shown below. We can notice that the routing algorithm make use of connections like `((((3) (0 1)) R1) (((2) (1 1)) L0))`, to compute a route.

```
((1 FRM1
  (((((4) (0 0 1)) L) (((3) (0 1)) L0))
    ((((3) (0 1)) R1) (((2) (1 1)) L0))
    ((((2) (1 1)) R0) (((1) (1 0)) L1))
    ((((1) (1 0)) R0) (((0) ((1 0) (0))) L))))

(2 FRM2
  (((((4) (1 0 1)) L) (((3) (0 1)) L1))
    ((((3) (0 1)) R0) (((2) (1 0)) L0))
    ((((2) (1 0)) R0) (((1) (0 0)) L1))
    ((((1) (0 0)) R1) (((0) ((0
 0) (1))) L)))))
```

6. Conclusion

In this paper we have described a methodology for the integration of formal methods in the verification of on-chip communication architectures. We have also shown that it is possible to take advantage of the graph theory basics to build formally connections of any network topology. We believe that we can apply the extended generic model to give a formal specification of any communication architectures.

A generic topology and extended routing components are designed to be included in the generic model GeNoC. We have developed the generic topology by identifying inherent properties of all topologies. These properties, which are called also constraints, have been validated using the ACL2 theorem proving environment. To achieve the routing extension, we have formalized the general common relation between topology and routing. In the case study, we specify and verify Delta multistage interconnection networks, as an instance of the extended generic model.

The framework presented in this paper opens promising trend for further development as complement to traditional verification techniques. We are currently investigated in the checking of the corresponding routing constraints and the integration of a pre-validated scheduling mechanism in order to validate the main GeNoC function.

7. References

[1] A. Jerraya, and W. Wolf, *Multiprocessor Systems-on-Chips*, Morgan Kaufmann Publishers, San Francisco, 2004.

[2] W. Wolf, ”The future of multiprocessor systems-on-chips”, Proc. of the 41st annual conference on Design automation, ACM Press, New York, 2004, pp. 681–685.

[3] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, “Floorplan-aware automated synthesis of bus-based communication architectures”, Proc. of the 42nd annual conference on Design automation, ACM Press, New York, 2005, pp. 565–570.

[4] L. Benini, and G. D. Micheli, “Networks on chips: A new SoC paradigm”, Computer, Vol. 35, N° 1, IEEE Computer Society Press, Los Alamitos, California, 2002, pp. 70–78.

[5] W. J. Dally, and B. Towles, “Route packets, not wires: on-chip interconnection networks”, Proc. of the 38th conference on Design automation, ACM Press, New York, USA, 2001, pp. 684–689.

[6] C. B. Stunkel, D. G. Shea, B. Aball, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker, “The SP2 High-Performance Switch”, IBM Systems Journal, Vol. 34, N° 2, IBM Corp., Riverton, USA, 1995, pp. 185–204.

[7] T. Cheung, and J. E. Smith, “A simulation study of the CRAY X-MP memory system”, IEEE Transactions on Computers, Vol. 35, N° 7, IEEE Computer Society, Washington, 1986, pp. 613–622.

[8] S. Duquennoy, S. Le Beux, P. Marquet, S. Meftali, and J-L. Dekeyser, “MpNOC Design: modeling and simulation”, In 15th IP based SOC Design Conference (IP--SoC 2006), 2006.

[9] C.P. Kruskal, and M. Snir, “The performance of multistage interconnection networks for multiprocessors”, IEEE Transactions on Computers, Vol.32 , N° 12, IEEE Computer Society, Washington, 1983, pp. 1091–1098.

[10] Y. Aydi, S. Meftali, M. Abid, and J-L. Dekeyser, “Design and Performance Evaluation of a Reconfigurable Delta MIN for MPSOC”, In 19th International Conference on Microelectronics (ICM ’07), 2007.

[11] T. Kropf, *Formal Hardware Verification - Methods and Systems in Comparison*, Springer-Verlag, London, 1997.

[12] Y. Aydi, S. Meftali, M. Abid, and J-L. Dekeyser, “Dynamicity Analysis of Delta MINs for MPSOC Architectures”, STA'07, 2007.
[13] T. Kropf, *Introduction to Formal Hardware Verification*, Springer Verlag, London, 1999.
[14] A. Roychoudhury, T. Mitra, and S.R. Karri, ”Using Formal Techniques to Debug the AMBA System-On-Chip Protocol”, Proc. of the conference on Design, Automation and Test in Europe, Vol.1, IEEE Computer Society, Washington, 2003, pp. 828-833.
[15] H. Amjad, “Verification of AMBA Using a Combination of Model Checking and Theorem Proving”, Electronic Notes in Theoretical Computer Science, Vol.145, 2006, pp. 45-61.
[16] B. Gebremichael, F. Vaandrager, M. Zhang, K. Goossens, E. Rijpkema, and A. Radulescu, “Deadlock Prevention in the Aethereal Protocol“, In D. Borrione and W. Paul, editors, Springer-Verlag, Germany, 2005, pp. 345-348.
[17] J. Schmaltz, and D. Borrione, “Towards a Formal Theory of Communication Architecture in the ACL2 Logic”, Proc. of 6th international workshop on the ACL2 theorem prover and its applications, ACM Press, New York, 2006, pp. 47- 56.
[18] M. Kaufmann, and J S. Moore, “ACL2: An industrial strength version of nqthm”, IEEE Transactions on Software Engineering, Vol. 23, N°4, IEEE Press, New York, 1996, pp. 23-34.
[19] D. Borrione, A. Helmy, and L. Pierre, “ACL2-based Verification of the Communications in the Hermes Network on Chip”, Proc. International Workshop on Symbolic Methods and Applications to Circuit Design (SMACD'06), 2006.
[20] C. Kruskal, “A unified theory of interconnection network”, Theoretical Computer Science, Vol.48, N°1, Elsevier Science Publishers Ltd., essex, 1986, pp. 75-94.
[21] D. Groth, and T. Skandier, *Network + Study Guide*, Sybex, San Francisco, 2005.
[22] M. Kaufmann, and J S. Moore, “Structured Theory Development for a Mechanized Logic”, J. Autom. Reasoning, Kluwer Academic Publishers, USA, 2001, pp. 161-203.