Power and Chip-Area Aware Network-on-Chip Modeling for System on Chip Simulation

Masoud Oveis-Gharan and Gul N. Khan
Electrical and Computer Engineering, Ryerson University
350 Victoria Street, Toronto
Ontario M5B 2K3 Canada
gnkhan@ee.ryerson.ca

ABSTRACT
In this paper, a Flexible And Accurate Network-On-chip Simulator (FAANOS) is introduced. NoC is a critical structure for system-on-chip design. We discuss the structure of its various components by presenting their details. We also provide an analytical methodology that employs the micro-architectural level of routers and links of NoC by considering their power and chip area requirements. We go through the structure of FAANOS when it is in switching mode and explain the transactional power estimation metric. To evaluate the effectiveness of an NoC system, an evaluation flow for early stage design and simulation of NoC is presented.

Categories and Subject Descriptors
B.4.4 [**Input/Output and Data Communications**]: Performance Analysis and Design Aids – *simulation, verification.*
C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors) – *interconnection architectures.*

General Terms
Design, Verification.

Keywords
Network-on-Chip, SystemC based Modeling, System-on-Chip Design.

1. INTRODUCTION
In VLSI technology, we recently have had a revolutionary development of Network-on-Chip (NoC) to deal with on-chip communication. In SoC design, there are a lot of challenges, and among them communication and synchronization between modules are prominent [8]. As a solution to these and other problems such as performance, chip area and power consumption, NoC structure has emerged. NoC replaces design-specific interconnection (buses, point-to-point ports, etc.) in SoC with a scalable and general purpose network, and it establishes a communication mechanism between NoC cores and modules [2]. For designers and architects of NoCs, obtaining an optimal NoC is a big challenge due to a number of constraints and objectives such as topology, switching method, routing techniques and traffic pattern that have impact on the power, area and performance of NoC. One of the best solutions to this challenge is to simulate NoC in a suitable environment. NoC simulators are dedicated test bed frameworks which serve a variety of NoC needs. They allow designers to test NoCs that might be difficult or expensive to emulate using real hardware. For example, simulating the effects of a sudden burst in a message flow will be difficult to investigate in hardware setup.

SIMUTools'14, March 17–19, 2014, Lisbon, Portugal.

Power dissipation in NoC circuits is generally classified into two sources: static (or leakage) and dynamic (or switching). The estimation of dynamic power dissipation of any chip design is a challenging task. Two main power models ranging from RTL power estimation tools [12] and early stage architectural power models have been proposed [14, 6]. For RTL power estimation, the power models can be obtained by synthesis and place/route of the RTL code. The generated library of the NoC components is then used during NoC topology synthesis. The shortcomings of this model are the requirement of having complete RTL code and slow simulation (in the range of hours). However, the architectural power model takes may only few seconds. This has encouraged us to choose the architectural power model to incorporate into our NoC simulation framework. This model allows NoC architects and us to estimate the power in transactional level or maximum condition. This feature of our NoC simulator is unique as compared with the past similar works reported.

We first develop a homogenous, general and accurate NoC simulation framework. The backbone of the Flexible And Accurate Network-On-chip Simulator (FAANOS) tool is built using SystemC, and all of its components are developed in our group. Moreover, the simulator considers different bit rates in the application-oriented workloads and supports them by producing appropriate output results. Our FAANOS simulator implements a complete and diverse routing mechanism in 2D topologies by using Source, Virtual-circuit, Odd-Even and LP routing techniques [9]. These mechanisms lead to flexibility and accurate results, and it provides an efficient NoC simulation and design environment. FAANOS employs an analytical method to estimate the power and area of NoCs that is a fast and accurate methodology. It allows experimenting with various options available at every stage of NoC design such as topology generation, switching technique, virtual channels and routing methods. FAANOS can be easily extended to include new topologies and routing algorithms. In terms of accuracy, we can argue that in our general NoC simulation environment the metric results tend to be accurate needed at early NoC design stage.

The outline of this paper is as follows. Section 2 provides a brief discussion on the related work. The infrastructure of NoC

Simutools 2014, March 17-19, Lisbon, Portugal

DOI 10.4108/icst.simutools.2014.254626

simulator is introduced in section 3. The transactional power estimation is presented in section 4. Then we propose an evaluation flow model for NoC systems in section 5 and follow it with the investigation of a test case in section 6. Finally, we provide conclusions in section 7.

2. PAST NOC SIMULATORS

Most of the existing NoC simulators consider only a few aspects of NoCs. Banerjee et al. have developed an RTL level power model for NoCs by first extracting the SPICE level net list from the layout and then integrating the characterized values into the VHDL based RTL design [1]. An accurate power characterization of a range of NoC routers was performed through RTL synthesis and place and route using standard ASIC tool flow by Synopsys [12]. Bona *et al.* first proposed an architecture level power model for interconnection networks, deriving its power estimates based on transistor count [5]. ORION is an early-stage architectural power model for NoCs that was originally proposed and released in 2002 [14]. It has since been widely used in academia and has been incorporated into industry tool chains such as Intel, AMD, IBM, and Free scale. Bhat also presented a methodology for automatically generating the energy models for on-chip communication infrastructure [3]. However, their focus is on bus and crossbar based communication for SoC systems.

In terms of NoC simulator development, a lot of research has been done on developing SystemC based simulation tools for analyzing different aspects of an NoC system. Among the recent past SystemC based NoC simulators, NIRGAM and NOSTRUM are the prominent one. NOSTRUM is a SystemC NoC simulator intending to develop an NoC architecture [11]. It mainly concentrates on the communication issues. NIRGAM is a SystemC based discrete event, cycle accurate simulator for NoC design [15]. It provides considerable support to examine different NoC designs in terms of routing algorithms and various topologies. It is capable of simulation on 2D regular mesh or Torus topologies using wormhole switching. The traffic generator in NIRGAM generates packets in constant and bursty bit rate. FAANOS covers all the functions of NIRGAM and NOSTRUM works and extends it with some new approaches. It considers different bit rates in the application-oriented workloads and supports them by producing appropriate output results. FAANOS implements a complete and diverse routing mechanism in 2D topologies by using the mechanisms such as Source, Virtual-Circuit, Odd-Even and LP routing [9]. These mechanisms lead to flexibility and simulation accurate results, and it is more supportive for users in providing efficient NoC design environment. FAANOS employs an analytical method to estimate the power and area of NoCs that is a fast and accurate methodology. It produces different power and area characteristics such as static, dynamic, architectural and transactional power estimation as well as link and router area that can completely describe the power and area behaviour of an NoC during early design stages.

3. FAANOS STRUCTURE

The internal structure of FAANOS simulator is shown in Figure 1. The blocks and arrows indicate the flow of data and information in the simulator. The first executive block is the User Interface. It gets NoC parameters in the form of various data values and text file listing various NoC cores. The User Interface converts the input data into core graph and core switch graph of NoC for visualization, and finally it generates a file which can be used for NoC simulation in the next block. The NoC Simulation block consists of all the hardware descriptions of NoC modules in the form of SystemC and these SystemC modules are utilized for producing output results.

The inputs to main functions of the NoC Simulator are the files, which are prepared by the user and the User Interface program, as well as libraries of power and area parameters. The NoC power and chip area details are embedded in the simulator. The NoC Simulation block uses these inputs to synthesize the final NoC structure and then start the simulation process. After the completion of a simulation run, it produces the area, power and performance results of the NoC.

3.1. Hardware Modeling of NoC Simulator

Our simulator is divided into a number of NoC modules that represent various areas of functionality of an NoC design. To better understand the structure of simulator, we start from a basic NoC design that is depicted in Figure 2. The design consists of a source module, a traffic generator, sink (receiver) and router modules. These four modules are connected by communication links. The source, sink and traffic generator modules play the role of SoC cores in FAANOS. We use only one traffic generator. However, the source, sink and router module can be more than one and are identified by unique ID numbers. Each module contains two basic elements such as port and process [17]. Ports allow communication among the modules. Processes are the main computational elements that execute concurrently.

The source module and traffic generator play the role of a source core. The source module uses a particular message structure, which provides the design access to packet definition and methods associated with the packet. A message consists of packets where a packet is formed by varying number of flits. The header flits are needed to route data from a source node to a sink node. The header flit has various routing information as depicted in Figure 3.

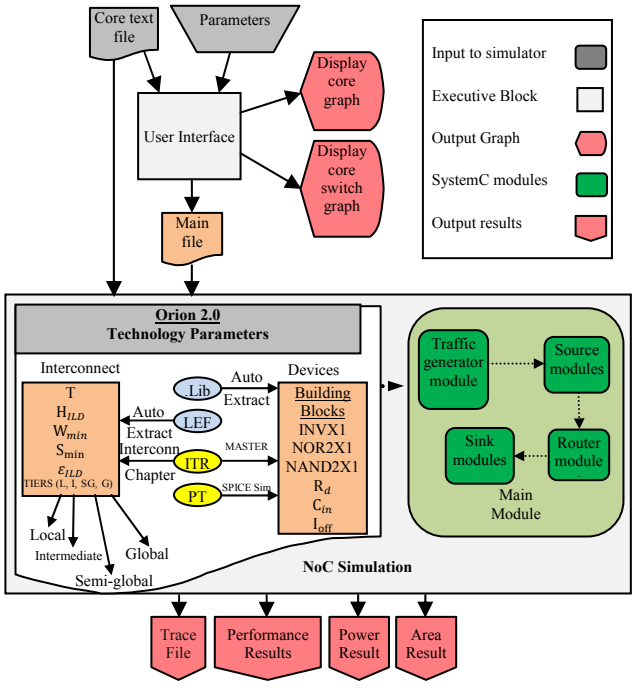


Figure 1: NoC simulator structure

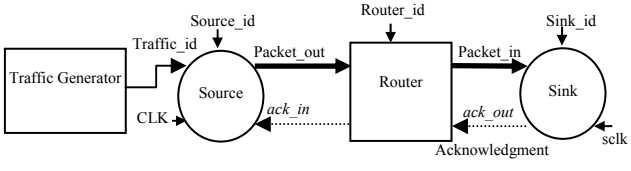
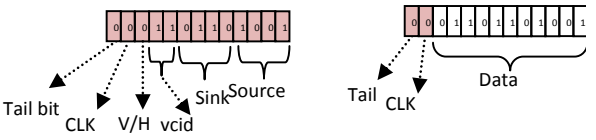


Figure 2: A basic NoC



(a) Header flit (b) Payload flit

Figure 3: Header and payload flit

The main purpose of the source process is to make packets depending on the packet specification and traffic pattern. The source process is invoked by a clock event. It calls two subroutines depending on the chosen traffic pattern. The pseudo code for the source process is given in Figure 4. The traffic generator is responsible for dictating the traffic pattern to the source modules. The pseudo code in Figure 5 shows the operation of the traffic generator process.

```
void source:: func()
{
  define and Initialize local variables;
  //continue till current time count is < SIM_NUM
  while( sim_count++ < SIM_NUM ) {
    wait(); //when current time is higher than
            //SIM_NUM*LOAD do nothing//
    if( sim_count >(SIM_NUM*LOAD))
        goto exclude;
    //read  the acknowledgment port of the router
    ack = (bool)ach_in.read();
    //when  the router does not accept packet
    if(ack) ++flt_drp; //when router accepts packet
   else {
     Handle the burst messages;
     Read the input port of traffic generator;
     Save it as destination ID;
     Generate packet related to destination ID;
     Send packet;
     Keep records of total time and number of packets
     Initialize for next clock cycle;
    }
  }
    sc_stop();
}
```

Figure 4: Pseudo-code of the source module

Two types of routers, regular and irregular are employed in our simulator. The regular router has five input ports and five output ports as shown in Figure 6. It is suitable for regular NoC topologies such as Mesh or Torus. It can execute all the routing techniques employed in FAANOS. However, the irregular router has flexible input and output ports depending on the application specific topology of the system. It is modeled to have a maximum of 16 input/output ports to realize and simulate irregular topologies. The router module accepts packets from the source (or other router modules) and passes them to the sink (or other router modules). When the router receives a packet, it puts the packet to a channel. The address of the channel is determined by the incoming packet. The router checks whether the channel is full or not. If the channel is full, the router sends back an acknowledgment (bit) to the source module requesting to hold any further packet transfer via that the channel. The router also waits for acknowledgment from the receiver module after it sends a packet to the receiver. The router consists of some lower level modules such as *FIFO, crossbar, arbiter* and *demux* which are connected by signals together as illustrated in Figure 6.

The sink module accepts packets from the router and keeps record of the number and time of incoming packets. It plays the role of a sink core. When the sink module successfully receives a packet, it sends an acknowledgment (bit) back to the router module.

```
// traffic_gen.cpp
void traffic_gen:: func()
{
  if(fixed) {     // traffic is fixed
    Send destination address of each source to
    related output port for fixed pattern;
  }
  if(UNIFORM) {   // traffic is uniform
    Randomly chooses a destination address from
    the current available destination addresses;
    Generate traffic pattern;
    Send the destination address of each source
    to related output port based on uniform pattern;
    }
   if(LOCALITY) { // traffic is locality
    Consider a source;
    Randomly Choose locality coefficient number;
    Determine neighbours related to the chosen
         locality coefficient number;
    Randomly choose a destination neighbor;
    Generate traffic pattern;
    Send destination address of each source to
    related output port for locality pattern;

    }
}
```

Figure 5: Pseudo-code of traffic generator

The *arbiter* module handles all the methods in a router such as routing and switching algorithms. When a packet is injected to an input port of router, it is directed by a demultiplexer to the first free virtual channel (FIFO buffer). The FIFO module sends the routing address of packet to the arbiter as a request event. When the arbiter detects that event, it reads the destination address and checks whether the output address is free or not. If it is free, then the packet is sent via that output port. The arbiter then disables a specific bit in the variable, *free_output* meaning that no data can be sent through that output port. This bit stays disable until the next clock event. When a clock event is invoked, the arbiter first checks that whether any output port gets free. If it is free, the arbiter enables the *free_output* bit related to that output port. Enabling this bit means that the related output port is ready to operate. The 2nd duty of the clock event is to pay attention to any unanswered requests. If there is any request, the arbiter checks its requested output port. If it is free, then packet can go through that port. If the output port is not available, the request will stay until the next clock event. Figure 7 lists the pseudo code of arbiter process.

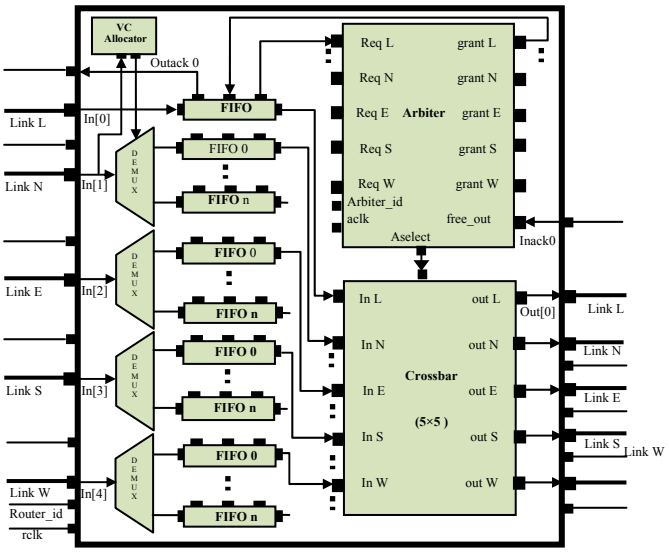


Figure 6: A 5×5 wormhole router

```
void arbiter :: a_func() {
 while( true )  {
  wait();
  initiate the local variable in each while loop;
  if ( aclk.event()) {    // Start of clock event
    Check any free output port ;
    Check any remaining request for output;
    If output of a reserved input is free, send flit;
    If line_probe, execute its function;
    If output of header-flit in VC is free, send flit;
  }
  else {
   if (req0.event()) { // Start of request events
    Read the header;
    If ODD_EVEN, execute its function;
    If line_probe, exdcute its function;   // [9]
    If the output port is free, start to send packet;
    Reserve output port until all flits pass;
   }
  }  // while
}        // end of a_func()
```

Figure 7: Pseudo-code of arbiter process

The demultiplexer module consists of a virtual channel allocator module and a switch module. It receives packet from a sender module via the input port of the router. The module (*demux*) directs the packet to a FIFO module depending on the *vcid* value of the header flit. Each input port of a regular router has at least a FIFO buffer module or a maximum of four FIFO modules. When the *demux* module directs a flit to the input port of FIFO module, the FIFO module writes the flit into the tail of its buffers. When the flit emerges at the head of FIFO, a request containing the route information is sent to the requesting port of arbiter module for the desired output port. After the arbiter module performs arbitration, it sends a grant signal to the *grant* port of FIFO module that leads to the activation of the read port of FIFO. The pseudo-code of Figure 8 depicts the functionality of FIFO process.

When a flit is injected to the input port of crossbar module, the packet from the input port (*config*) is sent from the router via the output port. The pseudo-code of Figure 9 is an event condition for the input *i0*.

```
void buf_fifo :: f_func(){
 while( true ){
  wait();
  if (wr.event()) { // start reading incoming packets
   read flit from the input port;
   store in the tail of FIFO ;
   send back the new condition of FIFO ;
   send a request to Arbiter ;
  }
  if (grant.event()) { // start sending the packets
   send flit to crossbar module;
   send back the new condition of FIFO;
  }
  if (bclk.event()){  // clock event
   send a request to arbiter module;
  }
 } // while
} // end of buf_fifo
```

Figure 8: Pseudo-code of FIFO process

The *main* function is the top-level entity that ties all the NoC modules together and provides the clock generation and tracing capabilities. The pseudo code of *main* simulator module is shown in Figure 9 where we instantiate each lower level modules and connect their ports with signals to create an NoC design for simulation.

3.2. Power Modeling

We propose an analytical method for estimating the power of NoC routers and interconnection links by changing the router architecture, ports and link widths. We use the architectural models of different NoC components as well as the parameters and analytical equations, which are established in these models to obtain the power estimates. Power dissipation in CMOS circuits can be classified into two sources: static (or leakage) and dynamic (or switching) power dissipation. The static power dissipation is currently one of the main factors limiting the performance of a computer system [7]. It is formulated as $P = I_{static} \cdot V_{dd}$ where I_{static} is the static current and V_{dd} is the supply voltage. The static power dissipation in each clock cycle is formulated as:

$P = I_{static} V_{dd}/F_{clk}$ where F_{clk} is the clock frequency

We can calculate I_{static} for each component of the NoC system depending on the architectural and technological parameters of each component of the system. However, V_{dd} and F_{clk} are the input parameters which should be specified by users in network simulation. For estimating leakage power, an architectural approach is proposed by Kahng et al [10] where the leakage current have almost a linear relationship with the transistor width and formulated by:

$$I_{leak}(i, s) = W(i, s) \cdot (I_{sub}(i, s) + I_{gate}(i, s)) \quad (1)$$

where

- I_{sub} and I_{gate} are sub-threshold and gate leakage currents per unit transistor width for a specific technology, respectively;
- W(i, s) refers to the effective transistor width of component *i* at state *s*.

The dynamic power is estimated based on switching power dissipation and it is formulated by:

$$P = \frac{1}{2}\alpha.C.V^2_{dd}.f_{clk} \quad (2)$$

```
// main_noc.cpp
int sc_main(int argc, char *argv[]){
 Define or Declare local signals, variables, clocks;
 Instantiate traffic-generatore, sources,
   sinks and routers;
 Connect traffic generatore's ports, sources' ports,
   sinks' ports and routers' ports to lacal signals;
 Trace file instractions;
 sc_start();         // start simulation
 Close trace files; // stop simulaton
 if (REG_TRAFFIC)    // regular traffic
  Calculate the performance, power and area metrics;
 if (IRREG_TRAFFIC) // irregular traffic
  Calculate the performance, power and area metrics;
}
```

Figure 9: Pseudo-code of main function

where

- α is the switching activity and C is switch capacitance;
- V_{dd} and f_{clk} are the supply voltage and clock frequency respectively.

Now by decomposing the NoC circuit into many equivalent RC circuits and using simple RC equations, we can estimate the total dynamic power.

3.3. Gate Level Modeling

We describe the gate level modeling of different NoC modules such as *arbiter, FIFO, crossbar* and *demux*. In spite of different routing strategies in the arbiter, three types of arbiters namely matrix, round robin, and queuing are modeled in FAANOS. Here, we just explain the matrix arbiter, which is introduced by Dally and Towles [8]. For an arbiter with *R* requesting entities, one can represent its priorities by an *R*×*R* matrix. With one in row *i* and column *j,* if a requester *i* has higher priority than another requester *j*, and zero otherwise. Then this method requires R(R −1)/2 matrix elements (flip-flops). Equation 3 represents the relation between the *n*th grant and the requests.

$$gr_n = req_n \cdot \sum_{i \neq n} (\overline{req_i} \cdot \overline{m_{i(n-1)}}) \quad (3)$$

where

- req_i is the i_{th} request and gr_n is the n_{th} grant;
- m_{ij} is the i_{th} row and j_{th} column element in the matrix.

Our FAANOS simulator models the crossbar and *demux* in terms of matrix and multi-tree. Figure 10 shows a gate level structure of normal matrix crossbar with *I* input ports, *O* output ports and *W* port width in bits. The connectors are shown as the transmission gate.

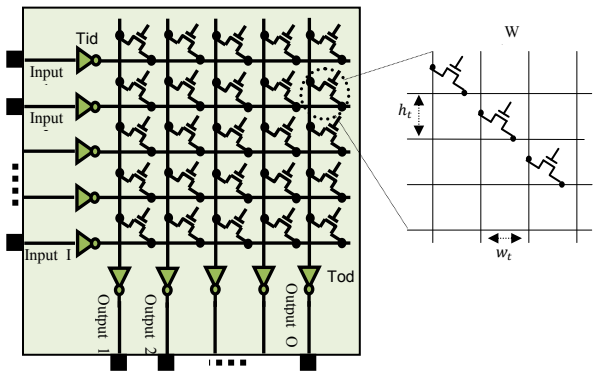


Figure 10: A typical matrix crossbar

The matrix crossbar consists of several horizontal input wires over vertical output wires, plus transmission gate or tri-state buffers at the cross-points. These tri-state buffers allow each input port to be electrically connected to any output port [13].

The FIFO buffer is modeled in two types of SRAM and register bases in our FAANOS simulator. Figure 11 shows the gate level structure of a normal SRAM.

3.4. Derivation of Gate Level Parameters

We need two characteristics I_{sub} and I_{gate} for each basic circuit components to estimate the static power. I_{sub} and I_{gate} for each component are per micron of gate width as listed in Table 1 for different input state ***s*** at 25°C and for a high *V*. These characteristics are derived from HSPICE and 65nm foundry SPICE model with corresponding technology parameters. FAANOS can calculate the I_{leak} for each module by employing Equation 2 as the structure of each NoC module is hierarchically composed from these circuit components.

Table 2 lists the capacitance notations needed for the basic circuit components used in the gate level models of NoC modules to estimate the dynamic power. The actual values of C_g, C_d and C_w are computed by CACTI [15]. Transistor sizes can be the user-input parameters or automatically determined by Orion 2.0 [10] with a set of default values from CACTI and applied with scaling factors from WATTCH [6]. Sizes of the driver transistors, e.g. crossbar input drivers are computed according to their load capacitance. For each component, we first describe its regular structure in terms of architectural and technological parameters. We then proceed with a detailed analysis and derive parameterized capacitance equations by taking into account both the gate and wire capacitances. The capacitance equations are then combined to estimate switching activity to determine energy consumption per component operation.

4. TRANSACTIONAL POWER

The transactional power estimation is the power consumption when an NoC is operated by a specific traffic pattern. In this case, we estimate maximum static power for all the components and dynamic power for only those components that switch during simulation. When data travel between NoC modules, each module keeps records of its transactional characteristics. These records facilitate the collection of transaction statistics and assist the integration of power results in FAANOS. When a simulation is stopped, the traversal energy (the energy consumed at each transaction in a module) related to each module is multiplied by the transactional record of that module and then accumulated, which result in the energy consumption of NoC design.

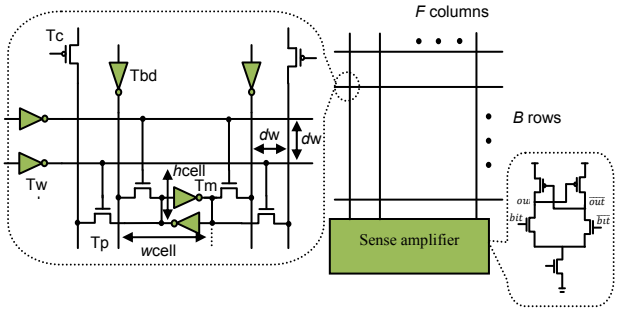


Figure 11: SRAM-based FIFO buffer with a read & write

Table 1: I_{sub} & I_{gate} for each basic Circuit Component

component	s	I_{sub} (A)	I_{gate} (A)
NMOS	0	1.097e-07	4.622e-09
PMOS	1	3.172e-07	3.291e-09
INV	0	1.097e-07	4.622e-09
	1	3.172e-07	3.291e-09
NAND2	00	7.098e-08	3.549e-09
	01	1.134e-07	5.103e-09
	10	1.342e-07	1.194e-08
	11	1.766e-07	1.625e-08
NOR2	00	1.971e-07	6.701e-09
	01	1.034e-07	4.343e-09
	10	1.412e-07	8.048e-09
	11	7.245e-08	6.448e-09

With the entrance of a flit, the router keeps a record of incoming flit in N_{link}. A router covers all the links events except

the links connected to the local sinks. Therefore, a router should also keep a record of incoming flit only for the exit link connected to a sink. Now, the dynamic energy for passing header flits inside a router during simulation can have the following equation.

$$E_{dyn\_router} = N_{\text{link}}.E_{link} + N_{\text{vc al}}.E_{vc\_al} + N_{\text{wr}}.E_{write} + N_{\text{read}}.E_{read} + N_{\text{arb}}.E_{arbiter} + N_{\text{crossbar}}.E_{crossbar} \quad (4)$$

where

- E_{link}, E_{vc_al}, $E_{arbiter}$ and $E_{crossbar}$ are the traversal energy of *link, VC_allocator, arbiter and crossbar* modules respectively when a transaction happens in a module.
- E_{write} *and* E_{read} are the traversal energies when a write and read transaction happens in a FIFO module.
- N_{link}, N_{vc_al}, $N_{arbiter}$ and $N_{crossbar}$ are the incoming records of *link, VC_allocator, arbiter and crossbar* modules respectively.
- N_{write} *and* N_{read} are the records when a write and read transaction happens in a FIFO module.

Table 2: Capacitance Notations

Notation	Description
Cg(T)	Gate capacitance of transistor or gate T
Cd(T)	Drain capacitance of transistor or gate T
Ca(T)	Only applicable if *T* is an inverter. *Ca(T)=Cg(T)+Cd(T)*
Cw(L)	Capacitance of metal wire of length *L*
Cin_cnt	Input node capacitance of a crossbar connector
Cout_cnt	Output node capacitance of a crossbar connector
Cctr_cnt	Control node capacitance of a crossbar connector
CFF	Switch capacitance of a flip-flop
CFC	Clock capacitance of a flip-flop

As the arbitration and VC (Virtual Channel) allocation happen only for header flits, the dynamic energy for passing the body flits in a router during the simulation is calculated by employing Equation 5.

$$E_{dyn\_router} = N_{\text{link}} \cdot E_{link} + N_{\text{wr}} \cdot E_{write} + N_{\text{read}} \cdot E_{read} + N_{\text{crossbar}} \cdot E_{crossbar} \quad (5)$$

Each router in the NoC accumulates the dynamic energy for the flits, which has passed through it. The following equations show the process of calculating the total power of NoC network, P_{network}.

$$E_{\text{dyn}} = \sum_{i=1}^{m \times n} E_{dyn\_router}(i)$$

$$P_{\text{dyn}} = \frac{E_{\text{dyn}}}{N} \times f_{clk}$$

$$P_{\text{static}} = (P_{\text{leak}}(\text{v\_al}) + P_{\text{leak}}(\text{FIFO}) + P_{\text{leak}}(\text{ar}) + P_{\text{leak}}(\text{cr}) + P_{\text{leak}}(\text{l}))\ m \times n$$

$$P_{\text{network}} = P_{\text{dyn}} + P_{\text{static}} \quad (6)$$

where

- $m \times n$, N, f_{clk} and E_{dyn} are the number of routers, the number of simulation cycle, the NoC frequency, the total dynamic energy of network during simulation time respectively;
- P_{dyn} and P_{static} are the total dynamic and static power of NoC network;
- P_{leak}(v_al), P_{leak} (FIFO), P_{leak} (ar), P_{leak} (cr)and P_{leak} (l) are the static power of *VC_allocator, FIFO, arbiter, crossbar and Link* modules respectively.

5. EVALUATION FLOW MODEL

We propose an evaluation flow model by which the effectiveness of an NoC system is evaluated in terms of performance, power and area metrics by using NoC simulator. One should first configure the network and traffic pattern and then apply different traffic patterns to evaluate an NoC as depicted in the flow chart of Figure 12. The NoC evaluation flow may be iterative until a user is satisfied. There are two loops in the flowchart. The first one is partially implemented in the User Interface program and the rest in the NoC Simulator. In this step, the user need to specify the configuration of NoC system in terms of size and kind of message, topology, switching and routing mechanism, FIFO buffer and virtual channel. These configurations are specified in terms of NoC parameters and a core text file. The parameters are requested via the console screen by the User Interface program that a user should include into the program. The core text file is a file produced from the core graph of NoC application. The output of User Interface is a file that is used as the main file of NoC Simulation program (Figure 1). In the second loop, the user cannot change the topology of NoC, but other configurations can be changed. These configurations are saved in the main SystemC file, *main_noc.cpp*. The user can change them and run the NoC Simulation iteratively to gain the best results of NoC.

6. TEST CASE STUDY

These analytical investigations demonstrate not only the capability of our simulator but also the challenges in response to satisfying the NoC constraints. This investigation is presented on 2D topology. We have selected the test cases based on several criteria such as the number of routers ranging from 9 to 36 and the number of virtual channels (one and four). The goal is to study the impact of NoC size and number of virtual channels on the power and area of NoC for torus topologies. Specifically, we are interested in the architectural and transactional power and area breakdown. We assumed a 90nm technology and a clock frequency of 1GHz. The area occupied by cores is fixed and assumed to be 9 mm^2. The other specifications of NoC are as follows.

- Number of simulation=1000 cycle
- Depth of FIFO = 2
- Traffic pattern = Uniform
- Size of each packet = 5 flits
- Routing strategy = XY mechanism
- Size of each flit =31 bits

The simulation results after synthesizing different torus NoCs are reported in Figures 13, 14 and 15. Each histogram is divided into two zones: one virtual channel (VC=1) and four virtual channels (VC=4). Each zone contains four bars and each bar is for different NoC sizes such as 3×3, 4×4, 5×5 and 6×6. We choose different NoCs to show the effect of NoC sizes on NoC metrics. Since the NoC size determines the number of routers in a 2D topology, NoC with size 3×3 has 9 routers and more for other mentioned NoCs. Therefore, we expect a variation with slope of 9, 16, 25, 36 (or 1, 1.8, 2.8, 4) between the metric results of these NoCs.

In Figure 14, it can be easily grasped that the router area is a function of NoC size in both VC=1 and VC=4 zones. It is logical because routers have fixed size (5×5) for torus topology, and the NoC size has a linear relation with the number of routers. Therefore, the slope of increase in the router area follows the NoC size in both zones of Figure 14. However, the slope of increasing for the link area is not the same as that of router area. This is because the area of link is a function of the total link length that is estimated based on the SoC area, and the SoC area consists of the area occupied by the cores and the routers.

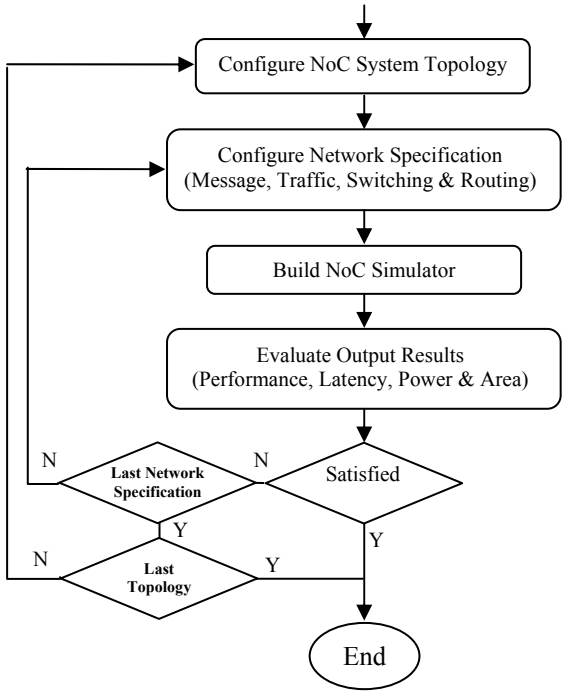


Figure 12: NoC evaluation flow

The area occupied by the routers is a function of the NoC size, but the area occupied by cores is assumed to be fixed and 9 mm^2. Due to this fixed area, the slope of increasing for the link area will not be the same as the router area in both VC=1 and VC=4 zones. We have also observed the same issue in the architectural power graph.

In Figure 15a, the router architectural power is a function of the NoC size for both VC=1 and VC=4 zones. It make sense as the router size (5×5) is fixed, and the NoC size has a linear relationship with the number of routers. Therefore, the increasing slope of the router architectural power follows the NoC size in both zones. However, the increasing slope of link architectural power is not similar to the router area. As mentioned before, this is because the cores area is fixed and it is not a function of the NoC size. In the case of transactional power graph, the behaviour of bars are different for VC=1 and VC=4 zones. In the case of VC=4 zone shown in Figure 15b, when the routers have 4 virtual channels (for a 2D topolgy), there is no contention in the NoCs and the throughputs are almost 100% as depicted in Figure 13.

The transactional power is a function of the rate of router and link transactions in NoC. The rate of router transaction when throughput is a function of NoC size. In other word, when NoC size increases, the number of cores will increase and more transactions occur in NoC, so when throughput is 100%, the router transactional power will be a function of the NoC size. However, the same issue as mentioned before is happen for link transactional power. The constant value of core size changes the corresponding relation between the link transactional power and the NoC size. In this case, the slope of increasing in transactional power bars resembles the architectural power in VC=4 zone of Figures 15a and 15b.

In the VC=1 zone of Figure 15b, there is no relationship between the transactional power and the NoC size. This can be justified by studying the throughput simulation results depicted in the bar graphs of Figure 13 where the average of throughput is less than 100%. In this case, we cannot estimate the behaviour of NoC transactional power. In other words, in a deterministic (XY) routing strategy, the higher throughput represents the higher rate of transaction that leads to larger amount of transactional power consumption by the NoC. For example, the 6×6 bar is less than the bars of other bars because the corresponding bar in Figure 13 is the least.

Two other points highlighted by the results shown in Figures 14 and 15 are the inconsiderable static power and considerable link power and area. The static power component in the architectural power is very small that can be ignored in our design. However, in the case of transactional power especially in the VC=1 zone, it can be a considerable amount in compared to dynamic power. Therefore, it needs to be available in NoC design. The link characteristic is important because sometimes it includes a significant amount of NoC power. For example, in Figure 15a, the link architectural power contributes over one-third of each bar representing the overall power.

By considering all the mentioned points, we can conclude that the architectural power consumption and the area occupied by the network are the functions of the structural specification of an NoC, which is the size and number of virtual channels used in 2D (Torus) topologies. However, NoC transactional power is the functions of the structural and transactional specifications of NoC including size, number of virtual channels and throughput. The transactional power is very close to realistic power of NoC that can be helpful to determine the bottleneck of NoC power, and the architectural power determines the maximum power that is consumed in the NoC. This will be very helpful in the early stage of NoC design. For example, in the case of four virtual channels (VC=4), which represents a guaranteed contention free NoC, the average area occupied by the routers is 9.1 times higher than the average area for one virtual channel (i.e. VC=1). However, the variation in architectural power is much lower. For example, the average power in the VC=4 zone is 1.7 times more than the average power in the VC=1 zone. This can provide an insight to the designer that by increasing virtual channels from one to four will result in more cost of chip area than the power.

7. CONCLUSIONS

The structure of FAANOS has been analyzed in this paper. We used SystemC, which is a C++ class library with some dedicated language constructs, to create cycle-accurate models of the hardware part and a user interface program. The hardware modeling of every NoC module was described in detail. We demonstrated the developing of our NoC simulator from a simple network to a more complex and complete NoC simulator. In terms of power and area, we have presented an analytical model for the power and IC area of different modules of an NoC such as FIFO buffer, arbiter, crossbar, and link. We have introduced different state of the art architectural model of each NoC module and then decomposed them into gate level and presented the technological and analytical parameters accompanied by the related formulation. We also described how the transactional power metric of an NoC are calculated in FAANOS. At the end, an evaluation flow model for early design of NoC has been presented.

ACKNOWLEDGMENTS

The authors acknowledge the financial support from NSERC and Ryerson University. The authors of this work would also like to thank Dept of Electrical and Computer Engineering, Ryerson University for providing partial financial support.

8. REFERENCES

[1] A. Banerjee, R. Mullins, and S. Moore. A power and energy exploration of Network-on-Chip architectures. In Proceedings *First International Symposium on Networks-on-Chip,* pages 163–172. Princeton New Jersey, May 2007.

[2] L. Benini, and G. De Micheli. Networks on Chips: A new SoC paradigm, *IEEE Computer,* 35(1):70–78, Jan. 2002.

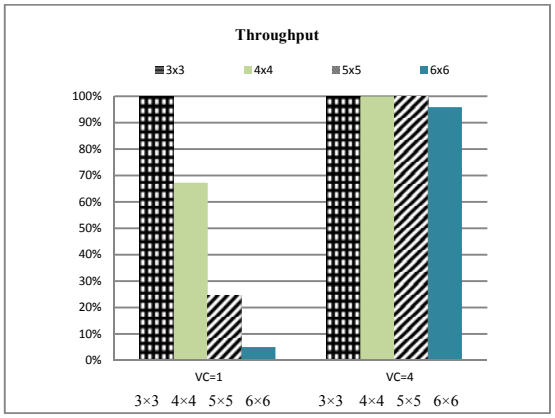


Figure 13: Synthesized % throughput for 3×3 - 6×6 Torus

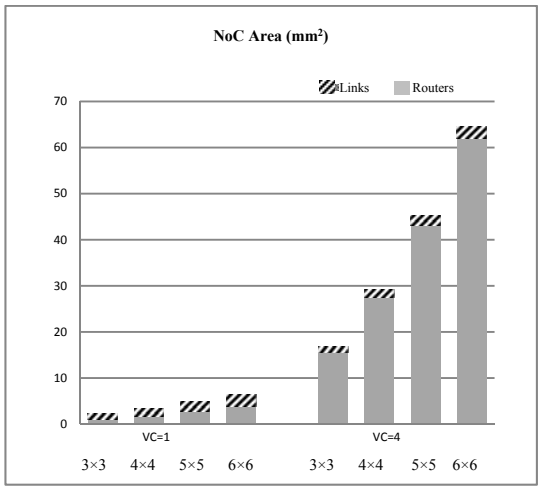


Figure 14: Synthesized chip area for 3×3 - 6×6 Torus

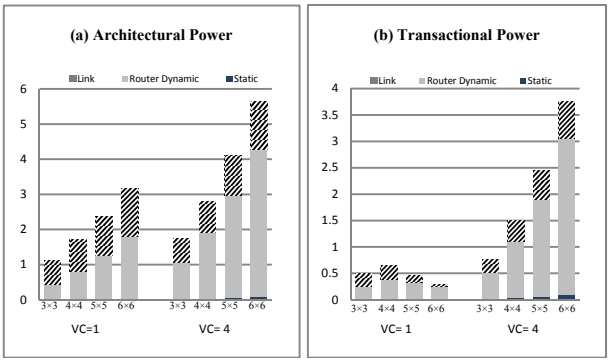


Figure 15: Synthesized power (watts) for 3×3 - 6×6 Torus

[3] S. Bhat. *Energy Models for Network-on-Chip Components*. M.S. Thesis, Dept. of Mathematics and Computer Science, Royal Institute of Technology Eindhoven, 2005.

[4] T. Bjerregaard, and S. Mahadevan. A survey of research and practices of Network-on-Chip. *ACM Computing Surveys,* 38(1):1–51, March 2006.

[5] A. Bona, V. Zaccaria, and R. Zafalon. System level power modeling and simulation of high-end industrial Network-on-Chip. In Proceedings *Design, Automation and Test in Europe Conference and Exhibition*, pages 318–323. Paris, France, February 2004.

[6] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In Proceedings *27th International Symposium on Computer Architecture,* pages 83–94. Vancouver Canada, June 2000.

[7] X. Chen, and L-S. Peh. Leakage power modeling and optimization in interconnection network. In Proceedings *International Symposium on Low Power Electronics and Design*, pages 90–95. Seoul, Korea, August 2003.

[8] W. J. Dally, and B. P. Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann publishers, San Fransisco, California, 2004.

[9] M. O. Gharan, and G. N. Khan. Flexible simulation and modeling for 2D topology NoC system design. In Proceedings *IEEE 24th Canadian Conference on Electrical and Computer Engineering*, pages 180–185. Niagara Falls, Canada, May 2011.

[10] A. B. Kahng, Bin Li, Li-Shiuan Peh, and K. Samadi. Orion 2.0: a fast and accurate NoC power and area model for early-stage design space exploration. In Proceedings *Design, Automation & Test in Europe Conference and Exhibition*, pages 423–428. Nice, France, April 2009.

[11] S. Penolazzi, and A. Jantsch. A high level power mdoel for the Nostrum NoC. In Proceedings *9th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools*, pages 673–676. Dubrovnik, Croatia, Aug/Sept 2006.

[12] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. *Technical Report HPL-2008-20*, HP Laboratories, April 2008.

[13] H. Wang, L-S. Peh, and S. Malik. "A power model for routers: Modeling Alpha 21364 and Infiniband routers," *IEEE Micro*, 23(1): 26–35, Jan/Feb 2003.

[14] H. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In Proceedings *35th Annual IEEE/ACM Int. Symposium on Microarchitecture*, pages 294–395. Istanbul, Turkey, 2002.

[15] W. Zhang, W. Wu, L. Zuo, and X. Peng. The buffer depth analysis of 2-Dimension mesh topology Network-on-Chip with Odd-Even routing algorithm. In Proceedings *International Conference Information Engineering and Computer Science*, pages 1–4. Wuhan, China, 2009.

[16] https://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/PowerCompiler.aspx, last accessed January 2014.

[17] www.accellera.org/downloads/standards/systemc, last accessed January 2014.