Scalable FPGA accelerator of the NRM algorithm for efficient stochastic simulation of large-scale biochemical reaction networks

Evangelos Koutsouradis, George Provelengios, Elias Kouskoumvekakis, Elias S. Manolakos

Department of Informatics and Telecommunications University of Athens, Greece
Email: {e.koutsouradis,eliasm}@di.uoa.gr

***Abstract*—Stochastic simulation of large-scale biochemical reaction networks, with thousands of reactions, is important for systems biology and medicine since it will enable the insilico experimentation with genome-scale reconstructed networks. FPGA-based stochastic simulation accelerators can exploit parallelism, but have been limited on the size of biomodels they can handle. We present a high performance scalable System on Chip architecture for implementing Gibson and Bruck's Next Reaction Method efficiently in reconfigurable hardware. Our MPSoC uses aggressive pipelining at the core level and also combines many cores into a Network on Chip to also execute in parallel stochastic repetitions of complex biomodels, each one with up to 4K reactions. The performance of our NRM core depends only on the average outdegree of the biomodel's Dependencies Graph (DG) and not on the number of DG nodes (reactions). By adding cores to the NoC, the system's performance scales linearly and reaches GCycles/sec levels. We show that a medium size FPGA running at ∼200 MHz deliver high speedup gains relative to a popular and efficient software simulator running on a very powerful workstation PC.**

Keywords: Systems Biology, Stochastic Simulation, Biochemical Reaction Networks, Next Reaction Method, SoC, FPGA, VHDL, Parallel Processing, Pipelining

I. Introduction

Systems Biology is a rapidly emerging multidisciplinary domain, on the cross road of biology, mathematics, computer science and engineering. It aims to study the dynamical behaviour of complex biological systems emanating from the interaction of their components. Biochemical reaction networks are commonly used to describe regulatory, signal transduction, or metabolic processes. The ever increasing complexity of biochemical reaction models, with thousands of biomolecular interactions, create the need for scalable PC accelerators to simulate them stochastically in reasonable time and space, at low cost and power consumption.

There are two main approaches for simulating biochemical reaction networks. The deterministic methods use ordinary differential equations (ODEs) to assess trends and steady state behavior of molecular species trajectories. On the other hand, Stochastic Simulation Algorithms (SSA) offer a more realistic alternative, since they mimic how reactions occur in nature in discrete steps and accounts for the stochasticity inherently present in biological systems. This approach works even in the low species counts regime where very interesting phenomena can be observed.

D.T. Gillespie introduced two popular stochastic simulation algorithms widely used today by software tools. Assuming a biomodel with m reaction channels, his Direct Method (DM) [1] has time complexity in $O(m)$. A later algorithm by Gillespie, the First Reaction Method (FRM) [2], has the same asymptotic complexity but is easier to parallelize for high performance. A remarkable more recent proposal by Gibson and Bruck [3], the *Next Reaction Method* (NRM), reduces complexity down to $O(log(m))$ and is preferred for large-scale models with very large m. However, it is a lot more difficult to parallelize since it has to manage a global data structure that all units of execution need to share.

Application-specific hardware architectures to accelerate the FRM SSA using FPGAs have been described in [4] and [5]. The implementation in [4] achieves a throughput of 10 Mega Reaction Cycles per sec (MRC/s) for small size biomodels. However it cannot efficiently handle large biomodels with more than 1K reaction channels. More recently, in [5] we have introduced a parallel FRM architecture that can handle biomodels with up to 4K reactions using up to 8 processors, implemented on a moderate size Xilinx Virtex 5 FPGA.

An FPGA based solution for accelerating the NRM-SSA is described in [6]. It can handle efficiently biomodels with up to 1K reactions. The key idea of the design is to separate the data structures (TPUs) from the computation units (TSUs), so that the architecture can scale up along with the biomodel's size. In addition throughput is boosted by allowing multiple simulation "threads" to proceed in parallel. Although this solution performs well for a small number of modules, as their number increases the interconnection network complexity limits severely the system's operating frequency. A more recent solution [7] proposed a massive parallel implementation of the NRM-SSA on a Virtex 6 FPGA. The key idea of this interesting design is to use as many parallel *Update Engines* as possible, each one responsible for just one reaction channel, fully pipelined and shared among different simulation instances. This system has the ability to process up to 1000 MSteps per second (also known as MCycles per second) for small size biomodels. However, its aggressive fine grain parallelism approach leads to inefficient FPGA resources utilization, severely limiting its scope to small biomodels with up to 64 reactions.

In this paper, we present a System on Chip parallel architecture that efficiently balances performance with hardware resources available on FPGAs. Our solution can scale to handle flexibly very large biomolecular networks (with up to at least $m = 4096$ reaction channels) while also achieving very high throughput e.g. ∼200 MCycles/sec per core on a Kindex 7 Xilinx FPGA. By using aggressive pipelining the main computation unit (the NRM core) exceeds the operating frequency of 200 MHz. In addition, by combining many cores into a Network on Chip system, we can run independent simulations

of biomodels or execute multiple stochastic repetitions of the same biomodel, and achieve linear performance scaling with the number of cores. The NRM core's design exploits the Dependencies Graph (DG) structure of biomodels, so that the achieved reaction cycle (step) latency depends only on the average outdegree (D_{aver}) of the graph and not on the number of its nodes (number of reactions, *m*). This leads to a reaction throughput per core that is independent of *m*, thus enabling the efficient simulation of very large biomodels with a medium size FPGAs.

The rest of the paper is organized as follows: in Section II we introduce stochastic simulation and present a summary of the NRM algorithm. In Section III we discuss our simulation framework and the interaction of the data structures required for an efficient hardware implementation. In Section IV we present an overview of the proposed many-core NRM NoC, the design of each core and how it operates during the simulation of a reaction cycle. In Section V, we present the NRM Processing Unit, the main computational module of a core, and how it was designed to optimize latency. In Section VI we present and discuss how the area utilization and performance of the NRM SoC correlate with the design parameters. We also report the achieved speed-up, as the number of cores increases, relative to COPASI [8], a popular software simulator.

II. Background

A stochastic chemical reactions network model is composed of *n* species $\{S_1...,S_n\}$ with initial population $\{X_1,...,X_n\}$ that can interact through *m* reaction channels $\{R_1,...,R_m\}$. To simplify the analysis we consider that all species are uniformly distributed within some volume Ω inside a cell of unit size. This assumption allows us to simplify calculations by ignoring the spatial effects existing in the real world. Let $X_i(t)$ be the population of species S_i at time t. The state of the network at time t is then $\boldsymbol{X}(t) = (X_1(t), X_2(t),, X_n(t))$ with initial conditions $\boldsymbol{X}_0(t) = \boldsymbol{x}_0$ at initial time $t = t_0$.

Stochastic simulation tracks the above state vector at appropriately chosen discrete time intervals, without explicitly solving the differential equations governing the underlying system dynamics to estimate the species trajectories. When a reaction R_μ occurs, the current state $\boldsymbol{x}$ is updated by a factor, so that $\boldsymbol{X}(t+\tau) = \boldsymbol{x} + \boldsymbol{\nu}_\mu$. The state update vector $\boldsymbol{\nu}_\mu$ is equal to $(\nu_{1\mu},..,\nu_{n\mu})$, where $\nu_{i\mu}$ represents the change in the molecular count of S_i due to the occurrence of reaction R_μ. Each reaction R_μ is also associated with a probability rate constant c_μ, which is proportional to the reaction rate constant k_μ and inversely proportional to the volume Ω (as shown in equations 7(a) and 7(b) of [2]) .

The probability that a randomly chosen combination of reactant species can interact and activate a reactions channel R_μ within the next infinitesimal time interval $[t, t+dt]$ is given by $c_\mu dt$. The propensity $a_\mu(\boldsymbol{x})$ of reaction channel R_μ at state $\boldsymbol{X}$ is calculated by multiplying the probability rate constant c_μ by the number of possible combinations of reactant molecules for R_μ in state $\boldsymbol{x}$. Thus for second order reactions with two reactant species S1, S2, it holds that:

$$\alpha_\mu = c_\mu * X_1 * X_2 \tag{1}$$

The model described above is a Markov process, where the next state depends only on the current one. Simulating this model yields the species trajectories of $\boldsymbol{X}(t)$. Gillespie's original DM SSA is based on the above formulation and speeds up the process by introducing a new function $p(\tau, \mu|x, t)$, which is the probability that R_μ is the first reaction to be activated in the system after the current time t and it occurs within the infinitesimal time interval $[t, t+\tau]$, given that the current state of the system is $\boldsymbol{X}(t) = \boldsymbol{x}$. This has the advantage that the simulation can advance from one time step to the next, without the need to simulate in-between times, at which no reaction occurs. So the update of the reactants and products happens in discrete amounts and species counts can be even very low, another notable advantage of SSAs over deterministic simulation methods.

While the DM SSA works fine for small biochemical networks, it is hard to parallelize and time consuming for medium to large size models. Concerned with these issues Gillespie introduced FRM-SSA. In this algorithm, a putative next reaction time τ_j is calculated for every reaction channel R_j. The reaction channel R_μ with the smallest putative time τ_μ is determined and is then "fired" at the end of the *Reactions Cycle* (RC). Since the calculation of each putative activation time τ_j can proceed independently of the others, the algorithm is a good candidate for massive parallelization.

Although the complexity of the FRM SSA is in $O(m)$, the algorithm can be accelerated if we realize that only a subset of propensities change as the result of a reaction event. Starting from this simple observation, Gibson and Bruck introduced the NRM SSA, with computational time complexity $O(log(m))$ [3] . We list below all steps of the algorithm:

1. Initialization:
   a. $t = t_0$ and $\boldsymbol{x} = \boldsymbol{x}_0$.
   b. Generate the Dependencies Graph (DG).
   c. Evaluate propensity functions $a_j(\boldsymbol{x})$ at state $\boldsymbol{x}$.
   d. Determine the time τ_j to the next R_j reaction:
   $$\tau_j = t + (1/\alpha_j(x)) * ln(1/r_j) \tag{2}$$
   where r_j is a unit uniform random number.
   e. Store the τ_j values in an indexed priority queue Q.
2. Let R_μ be the reaction with τ_μ is smallest stored in Q.
3. Determine the new state after Reaction R_μ: $t = \tau_\mu$ and $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{\nu}_\mu$, where $\boldsymbol{\nu}_\mu$ is the state change vector for R_μ.
4. For each edge (μ, α) in the Dependency Graph DG:
   - If $\alpha = \mu$, set
   $$\tau_\mu = (1/\alpha_\mu(x)) * ln(1/r_\mu) + t \tag{3}$$
   - If $\alpha \neq \mu$, set
   $$\tau_\alpha = (\alpha_{\alpha,old}/\alpha_\alpha) * (\tau_{\alpha,old} - t) + t \tag{4}$$
5. If t is greater than the desired simulation time, halt.
6. Record $(\boldsymbol{x}, t)$ and go to step 2.

The NRM algorithm comprises two distinct phases. The first phase includes initialization steps (1a) to (1e) performed once during each simulation repetition to prepare all the needed data structures. The second phase includes steps (2) to (6) and will be referred to as a *Reactions Cycle* (RC). In every RC, the NRM uses formula (3) to re-calculate the τ_μ of the (*winner reaction*) (R_μ) and formula (4) to re-normalize the putative times τ_α of every *dependent reaction* (R_α).

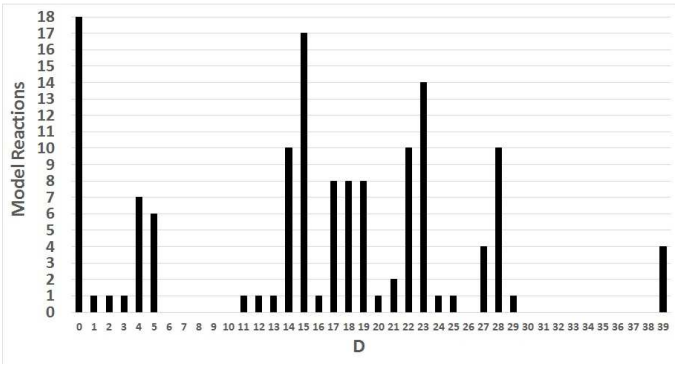

Fig. 1. Histogram of dependent reactions for the ASYN biomodel [9] [12]. The y-axis is the number of reactions in the model with the same D value.

According to equation (1), the propensity α_j of a reaction depends on the quantities of the reactant species. If these quantities change, as a result of another reaction R_μ, then only the affected propensities α_j should be re-calculated along with the affected τ_α. Therefore, when a reactant or a product species of R_μ is a reactant of another channel R_j, we say that these R_j reactions depend on R_μ and call them *dependent reactions* (R_α).

The NRM represents dependencies using a Dependencies Graph (DG). In the DG, there is a vertex for each reaction R_j and an arc (j, k), from reaction R_j to every dependent reaction R_k. So, assuming that *d* is the total number of affected reactions during an RC, there are $D = d - 1$ edges towards other reactions R_α which depend on R_μ (outdegree of node R_μ). Finally, based on the NRM, equation (3) re-calculates the time τ_μ of R_μ and (4) re-normalizes the time τ_α of every R_α.

If we analyze a real biomodel and the dependencies among its reactions, we will most definitely observe that each reaction has a different *D* i.e. outdegree of DG nodes is not the same. Let us consider for example the ASYN biomodel we have introduced in [9] [12], to simulate how Alpha-synuclein's (ASYN) oligomerization disturbs the homeostasis of dopaminergic neurons, a mechanism believed to contribute to the onset of Parkinson's disease. The biomodel has *m = 136* reactions and *n = 90* species. In Figure 1 we show the histogram of *D*. We observe that there are 18 reactions with *D = 0*, 4 reactions with *D = 39* and the mean value of is $D_{aver} = 16$.

III. Simulation Flow

A. Simulation Framework

The stochastic simulation framework we have developed includes a complete end-to-end simulation flow summarized in Figure 2. The simulator's primary input is the biomodel file supplied by the end-user in standard SBML format [10]. We have developed a parser (in C++) that can read the SBML file and extracts all essential information that is stored in three binary files: The *st_table.bin* containing the initial values of all species (*n*), the *rt_table.bin* describing all reactions (*m*), and the *dg_table.bin* capturing the inter-reaction dependencies. Each file is used to initialize a corresponding data structure, as discussed in Section III-B. Moreover, the user specifies simulation parameters, such as the total simulation time (T_{sim}), the output results sampling (reporting) period (T_{sam}) and the desired number of repetitions (R) of a model's simulation, which are stored in the *parameters.bin* file.

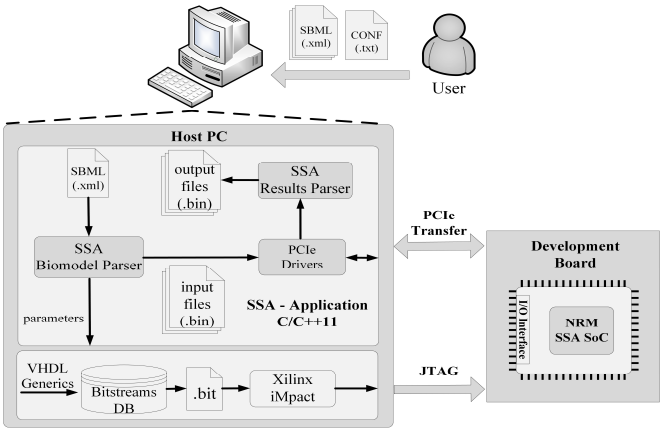

Fig. 2. The simulation flow including the Host-PC and the FPGA device.

The complete NRM SoC has been described as a parametric IP core in VHDL. The top-level parallel architecture is a Network on Chip (NoC) with *C* independent NRM cores, each using a different pseudo-random number generator and being responsible to perform *R* repetitions of a model's stochastic simulation. We have used the IP core to build a library of SoC bitstreams for many combinations of the design parameters *C*, *m*, *n* and D_{aver}. We have experimented with NRM SoCs containing up to $C = 16$ cores. Moreover, the biomodels can have up to 3^{rd} order reactions, with *m* and *n* values that vary from 256 to 4096 and D_{aver} from 4 to 128. Upon parsing the SBML model file, we can select the most appropriate already synthesized SoC bitstream in the library to program the FPGA.

The stochastic simulation is initialized by sending from the host PC all of the above input files through the PCIe component of the NRM SoC. During each sampling period, every NRM core sends back to the host PC the state vector of the simulation it performs. When the simulation completes, all output results are written into a binary file. Subsequently, a simulation output parser tool analyzes this file and produces a separate results file for each simulation repetition (run) performed by a processing core, including all species counts at each reporting period, as well as aggregate species statistics, such as the minimum, maximum, average population values over all repetitions.

B. Data Structures

Before the execution of a simulation, the *Species Table* (ST), the *Reactions Table* (RT) and the *Dependencies Graph* (DG) are the three important data structures that should be properly initialized. Figure 3 illustrates how these data structures cooperate to provide the necessary data, given the winner reaction R_μ. The ST contains the initial counts of all species and is a simple table with *n* rows (depth) and 32-bit width. The RT stores all of the reaction channel data. It has *m* rows (depth) and each row includes 8 pointers to the ST table (3 for the reactant and 5 for the product species), the reaction rate constant (c_μ) and the state change vector ($\boldsymbol{\nu}_\mu$). Lastly, the DG contains the Dependency Graph of the model. It has depth *m*, width $(D_{aver} + 3) * log(m)$ bits and is organized as two distinct parts. Each row of the first part is associated with a reaction and stores the number of its dependent reactions (*D*) along with a pair of elements *(i,j)* that form as a directory index to the second part. The second part is a table of pointers,

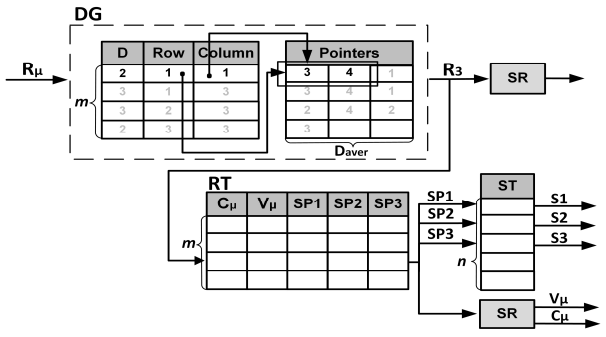

Fig. 3. The interaction between the data structures.

where each pointer holds an address to a reaction of the RT, depending on R_μ. Given the winner reaction R_μ (it is R_1 in the example of Figure 3) we read the directory index (first part) and then extract *D* pointers (second part), starting from the location (row, column) where the directory index points to (i.e. location *(1,1)* in our example). Since *D* differs across reactions, these pointers are not necessarily aligned and thus the dependencies of a reaction may occupy more than one line in the second part of the DG. This allows us to efficiently store more pointers in less memory and handle models of arbitrary dependent reaction profiles (e.g. see the ASYN biomodel of Figure 1).

At the beginning of a new reaction cycle, the hardware unit designated to handle the DG has knowledge of the R_μ information. Therefore, it can start exporting its dependent reactions, the R_α's, the one after the other in a steaming, fully pipelined fashion. For each R_α, the RT handling unit forwards its reactant species pointers (SP) to the ST. Subsequently, each dependent reaction's address and reactant species counts are forwarded to the processing element of the NRM core where only the necessary propensity calculations are performed. This interaction of data structures is shown for $R_\mu = R_1$ and $Ra = R_3$ (the first dependent reaction of R_1) in Figure 3.

IV. NRM NoC and Core Architecture

A simplified version of the NRM NoC architecture is shown in Figure 4. The architecture consists of *C* independent NRM cores connected in a star topology. The cores communicate with the PC through a custom switch, using round robin arbitration and the AXI4 Stream Interface protocol that is based on a Master/Slave communication model.

Each NRM core can execute *R* stochastic simulation repetitions of a biomolecular reaction network. Figure 5 provides a simplified block diagram of the NRM core consisting of the following six units: (1) the *NRM Processing Unit* (NPU), (2) *the Species Time Update Unit* (STU) which updates the molecular species populations (stored in ST), the values of the current simulation time (t_{sim}) and the last sampling (reporting) time (t_{sam}), (3) the *Simulation Control Unit* (SCU) which commands the simulation flow and the transmission of results back to the host PC, (4) the *Species Table Initialize Memory System* (STI MS) storing the initial counts of the *n* species, (5) the *Input Interface Unit* (IIS) receiving the simulation data, and (6) the *Output Interface Unit* (OIS) responsible for sending the simulation results back to the host PC.

The NPU is the most important component of the NRM core. It performs all NRM computation steps except from the

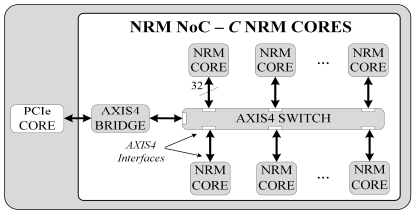

Fig. 4. The NRM Network on Chip top level architecture.

species updating step. The main units of NPU are the three memory subsystems (DG, RT and ST) storing and managing the data structures (DG, RT and ST respectively) in the available FPGA BRAMs and the *Processing Element* (PE) that performs the arithmetic computations. The PE unit consists of three subunits: The *Comparator Tree* (CoT), which implements the parallel hardware version of the NRM's Indexed Priority Queue (IPQ), *table T*, a memory unit which stores all the times τ_j and feeds the CoT, and the *Recalculation Unit* (RU), which re-calculates the τ_j of the reactions affected by the current winner reaction R_μ, based on equations (3) and (4) of the algorithm.

The NRM core operates in three distinct phases : *initialization*, *computation* and *updating*. Figure 5 shows an example of those phases during a reactions cycle. The initialization phase has two stages, 1.a and 1.b. During stage 1.a (signaled by `Init_a`) the IIS receives simulation data from the host PC and initializes the ST table of STI MS. It also initializes the RT tables and the NPU's pseudo-random number generators (RNGs), with their initial seeds. During stage 1.b (signaled by `Init_b`) the STI MS uses the saved data in order to initialize the NPU's ST table. Stage 1.a occurs only at the beginning of the simulation while 1.b is repeated at the beginning of each stochastic simulation repetition (if $R > 1$). These two stages are combined through a 2x1 multiplexer (see MUX (1) in Figure 5). The SCU asserts the signal `Init_Rep` to trigger stage 1.b if the number of completed repetitions is less than the desired value (*R*).

During the computation phase 2, the NPU performs steps 4 and 5 of the NRM and updates the putative times τ_j of the affected reactions. The SCU asserts the signal `Start Sim` to trigger phase 2 if the current simulation time has not been reached the desired simulation time set by the user ($t_{sim} < T_{sim}$).

Subsequently, during the updating phase 3, the STU calculates the new molecular species populations according to stoichiometry vector $\boldsymbol{\nu}_\mu$ of the R_μ and the new values of t_{sim} and t_{sam}. After the species updating, the core moves again to phase 2 and the SCU starts a new reaction cycle if $t_{sim} < T_{sim}$. During a reactions cycle the core uses a 2x1 multiplexer (see MUX2x1 (3)) to connect the STU's output to the rest of the units.

The SCU commands the whole simulation flow. The end of a stochastic repetition is signaled by the condition $t_{sim} \geq T_{sim}$. At this point the SCU checks if the counted repetitions reached the desired limit (*R*). If so, the SCU terminates the simulation, otherwise it triggers a new initialization phase 1.b through the signal `Init_Rep`. The SCU is also responsible of starting the transmission of simulation results. After every reaction cycle, the SCU checks if $_{tsim} \geq t_{sam}$. If so, it triggers

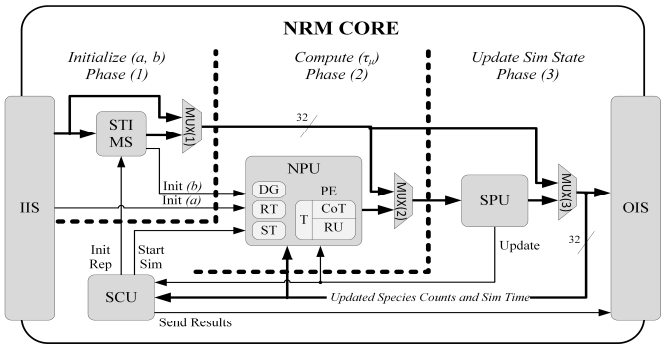

Fig. 5. The NRM core architecture overview.

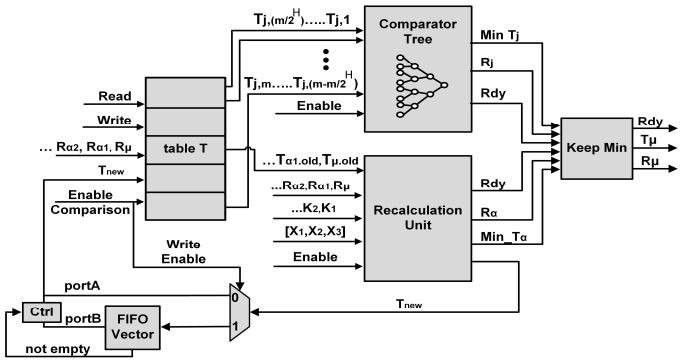

Fig. 6. The NRM Process Element architecture.

the OIS unit (through the signal `Send Results`), which in turn starts transmitting the simulation results back to the host PC.

V. NRM Processing Element Architecture

The NRM *Processing Element* (PE) is the main computation unit of the NRM core. As mentioned before, the PE receives information associated with the winner reaction, recalculates or re-normalizes the putative times of the dependent reactions and in parallel performs all required comparisons in order to determine at the end the next winner reaction.

As Figure 6 shows, the PE consists of three main units: *table T*, a memory unit that stores all the reaction times τ_j, the *Comparator Tree* (CoT), responsible to determine the reaction with the minimum time τ_j stored in *table T*, and the *Recalculation Unit* (RU), which re-calculates the winner reaction's time (τ_μ) using equation (3) and re-normalizes the dependent reactions time (τ_α) using equation (4) of the NRM.

As Figure 6 shows, table T is the entry unit of the PE, while the RU and the CoT are running in parallel. At the beginning of an RC, the PE receives the necessary data of all the affected reactions. At the same time, the RU starts executing calculations and "read-mark" the t_{old} time values stored in table T (the "read-mark" operation will be explained later in this section). When the RU finishes the "read-mark" operation, table T starts feeding the CoT with τ_j. Considering the fact that the height of the CoT depends on *m* and is configured to minimize the comparison time, the CoT is able to finish before the RU has executed all its computations. At the end, when the RU finishes, the results of the CoT and the RU are compared and the next winner reaction is determined.

A. The Comparator Tree and table T units

The CoT is a scalable binary tree of comparators, with height *H*, as a generic parameter in its VHDL description. To maximize the parallel export of data, table T is split into 2^H parts, as many as the leaf nodes of the CoT. Since *m* is the total number of reactions, each part of table T stores $m/2^H$ reaction times. Due to the fact that all BRAMs are dual ported memories, on every clock cycle table T sends $2^{(H+1)}$ data words to the leaf nodes and $m/2^{(H+1)}$ clock cycles are required to feed the CoT with the whole table T. Finally, after $L_{CoT} = m/2^{(H+1)} + H + 1$ clock cycles, all data have reached the root node, where the Keep Min unit compares the CoTs results and stores the reaction with the overall minimum τ_j.

In addition, table T, as a data structure, supports read and write functions. These allow the Recalculation Unit (RU) to read the old reaction times (τ_{old}) and write the updated values (τ_{new}). Every time the RU reads a time value from table T, at the same time it sets the value of this reaction to the maximum floating-point number in order to prevent the reactions that are not updated from reaching the root node. To perform this "read-mark" operation, we configured all the dual port BRAMs to "Read First" mode and so we can "Write after Read" using only one port on every clock cycle. In this way, the RU can "read-mark" the τ_{old} from table T using the first port and write the updated τ_{new} using the second port. Finally, in order to avoid any conflict, while the CoT is enabled, table T disables both read and write functions.

B. The Recalculation Unit

The Recalculation Unit (RU) performs all computation steps of the NRM. In order to implement equations (3) and (4), it includes several floating point computation modules and a local table to store the old reaction propensities. By observing that both equations contain almost the same calculations, differing only on their inputs, we implemented a pipelined datapath consisting of three parallel branches feeding a common branch, as illustrated in Figure 7. Each branch executes one or more computation steps and the common branch combines the results before calculating a τ_{new}. In order to activate the correct branches, during each reaction cycle, the NRM core forwards to the RU the data of the winner reaction and subsequently its dependent reactions.

We managed to set the latency of the modules in such a way that all branches are synchronized and produce results at the same time. Branch A includes the Propensity Unit and a division module. The Propensity Unit needs 14 clock cycle (cc) to calculate the α_j of a reaction and the division module needs 17 cc first to execute $(1/\alpha_\alpha)$ for the winner reaction and then $(\alpha_{\alpha,old}/\alpha_\alpha)$ for every dependent reaction. Branch B calculates the difference between the current simulation time t_{sim} and $\tau_{\alpha,old}$. Although this subtraction requires 11 cc, a shift register (SR) extends the latency of branch B to 31 cc. Finally, branch C computes the logarithm of a unit pseudo-random number. In case we want to re-calculate τ_μ based on (3), branch C is activated. In case of a dependent reaction, if the $\alpha_{\alpha,old} \neq 0$ then branch C remains inactive. However, if $\alpha_{\alpha,old} = 0$, branches A and C are activated in a proper way as to re-calculate τ_α based on (3) and not based on (4). So, branch C needs 4 cc to examine the $\alpha_{\alpha,old}$, 1 cc to enable the

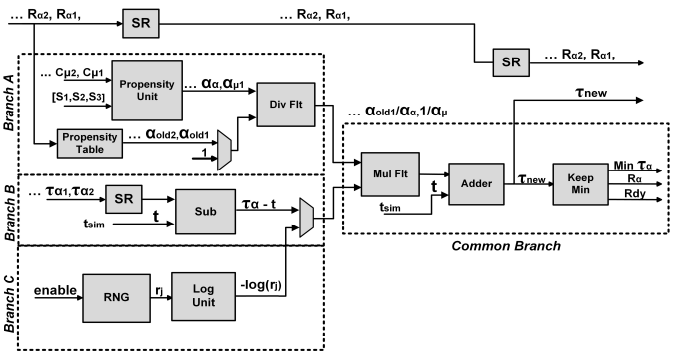

Fig. 7. The Recalculation Unit datapath.

Mersenne Twister unit and produce a pseudo-random number, 6cc to convert the result to a uniform number and 20 cc to compute the logarithm. Thus, Branch C has a latency of 31 cc in total.

At the end, the Common branch combines the results of branches A and C in order to compute equation (3), which re-calculates the τ_μ. Depending on $\alpha_{\alpha,old}$, the Common branch combines the results of all branches, so if $\alpha_{\alpha,old} \neq 0$ it performs (4) to re-normalize τ_α else if $\alpha_{\alpha,old} = 0$ it performs (3) to re-calculate τ_α. Overall, the Common branch has a latency of 11 cc..

At the end, all results go through the Keep Min unit, which compares and keeps the reaction with the smallest updated time. The RU has latency $L_{RU} = 42$ cc to update the time of an affected reaction. In addition, $D+1$ more cc are needed to determine the smallest update time τ_α.

C. Datapath and Latency Analysis

The NRM Processing Element works in two modes. The *initialization* mode is activated once, at the beginning of every simulation repetition, and aims to initialize the table T and the propensity table. In this mode, the RU calculates and stores a putative time τ_j for all the reactions of the model, the CoT remains inactive, and at the end the first winner reaction is determined. In the second mode (*running sim*) the PE executes all required computations to determine the next winner reaction of each RC, based on steps (4)-(5) of the NRM.

As mentioned before, the RU "read-marks" all the τ_{old} times from table T. When the RU produces a τ_{new}, it forwards the updated value to table T. If the CoT is not enabled, then the RU has access to table T and uses one port to update the marked value. However, if the CoT is enabled, the RU has no access to table T and thus we use a distributed FIFO to temporary store the results of the RU. In order to avoid overflow, the FIFO's depth must be equal to the maximum number of results that RU can produce, while the CoT is enabled, i.e. equal to the RU's pipeline depth. Finally, when table T becomes available again, all of the data from the FIFO are forwarded to it, while the core enters the species updating phase.

In conclusion, we have managed to design a high performance and fully pipelined datapath, with a latency that depends only on parameter *D*. Including all required computation steps and a few control steps, the PE latency is $L_{PE} = max(L_{RU}, L_{CoT})+D+1+6 = L_{RU}+D+7 = 49+D$. So, as the biomodel's complexity *m* is increasing, L_{PE} is not affected and the performance of the PE remains unchanged. Finally, as Figure 8 illustrates, the NRM core needs $L_{RC} = 92 + D$ cc to complete all phases of a reaction cycle (Reaction Cycle Latency). Note that L_{RC} depends on the number of dependent reactions (*D*) of the winner reaction and not on *m*. This is due to the design of a high performance computation unit, the NRM Processing Element, and the latency hiding methods we used for the CoT, in order to avoid extra latency while determining the next winner reaction.

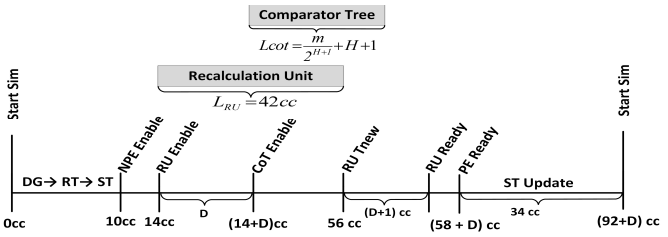

Fig. 8. Reaction Cycle Latency (L_{RC}) analysis.

VI. Area and Performance Evaluation

In this section we present and discuss how the area and performance of the NRM SoC correlates with the design parameters. We first focus on FPGA resources utilization and operating frequency. Then, we present the performance evaluation of our NRM SoC using a benchmark biomodel with complexity that can scale. Finally, we analyze its performance while simulating a typical model from the EBI Biomodels Data Base [11]. In order to do so, we measured the achieved speed-up as the number of cores increase, relatively to COPASI [8] a popular and efficient software simulator.

Our NRM SoC design was implemented on a Xilinx Kintex-7 board containing a medium size XC7K325T-2 FPGA with 203800 LUTs, 445 BRAMs (445x36 Kb) and 840 DSPs. The NRM SoC architecture was captured as an IP in parametric VHDL and uses several floating-point IP cores. All computation modules and data structures were implemented using the available FGPA resources (DSPs and BRAMs) and use single-precision floating-point format (32 bit, IEEE - 754).

Finally, for software simulations we employed COPASI [8], a popular and efficient single threaded software simulator, developed using C++. We run COPASI on a very powerful workstation PC with an Intel Core i7 4790K CPU running at 4.4GHz with 32 GB of RAM under GNU/Linux O/S.

A. Resources Utilization

Figure 9 summarizes the resources utilization (post place and route) and the max. operating frequency for several configurations of the parametric NRM SoC. All of them use one core, but differ in terms of the maximum number of reactions *(m)* and the average number of dependent reactions (D_{aver}) they can handle. We chose to report the total LUTs (left axis) and BRAMs (right axis) since these are the resources mostly affected by the problem size parameters of interest. Some FPGA resources are not fully utilized (e.g. DSPs) or change in a similar way as the LUTs (e.g Slices and their associated registers).

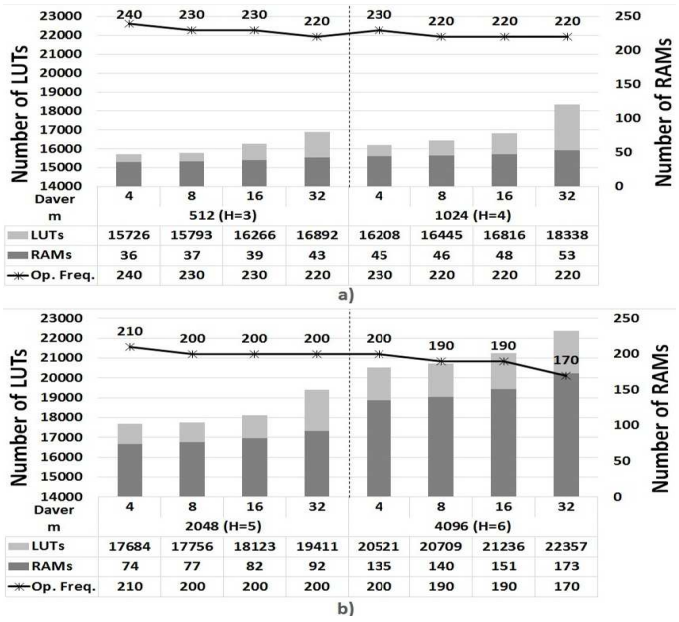

Fig. 9. Area Utilization (LUTs Left Axis - BRAMs right Axis) and operating frequency when Daver = 4, 8, 16, 32 and a) m = 512, 1024, b) m = 2048, 4096

The NRM SoC contains several data structures needed to store the simulation's input data. Most of them are simple structures with depth *m* and 32-bit width. But, there are two special memory structures, the DG and table T of the NRM PE that significantly affect the total number of BRAMs used by the design. In our implementation we handle the DG as a table whose depth is always *m* but the width is variable and equal to $(D_{aver} + 3) * log(m)$ bits. In order to achieve the required variable width, FPGA synthesis tools combine as many BRAM blocks as needed and manage them as a unified table. Unfortunately, as the number of combined BRAM blocks increases, this may complicate the FPGA routing and affect the operating frequency. As we see in Figure 9, for the same *m* the maximal operating frequency drops by 10-30 MHz, as D_{aver} increases between SoC configurations emanating from the same IP core using FPGA synthesis tools.

On the other hand, the total number of BRAMs required for table T depends on the height *H* of the CoT unit. Considering the CoT's architecture described in Section V, table T is divided into 2^H parts, as many as the leaf nodes of the tree, where each part reserves a BRAM of depth 1024 words and a fixed 32-bit width. In order to keep low the number of clock cycles required by the CoT to determine the next winner reaction, as *m* is increasing so does the height *H* of the CoT and therefore table T is split into more parts of the same size. As a result, every time we double *m*, the height of the CoT is increasing by one level and thus table T requires twice as many BRAMs. Moreover, the CoT demands a significant number of LUTs and every time its height increases by one level, twice as many comparators are needed, leading to the consumption of even more LUTs.

In conclusion, the area scalability of the NRM SoC architecture is an important issue that demands special attention in order not to reduce the operating frequency, while utilizing the minimum amount of resources. On the specific FPGA device we used, BRAMs is the limiting resource and so we can fit a network with up to *C=16* NRM cores, configured for medium size bio-chemical reaction networks ($n = m = 256, H = 2, D_{aver} = 16$), or up to *C=2* NRM cores, configured for very large biomodels ($m = n = 4096, H = 6, Daver = 96$). Moreover, the operating frequency can be improved by performing manual FPGA floor-planning and more cores can be added by using a larger Virtex-7 family FPGA.

TABLE I. PERFORMANCE EVALUATION - LCS MODELS

	NRM SoC - xc7k325t-2ffg			
m (Number of Reactions)	T_{Real} (Seconds)	Reactions Cycles (Millions)	Reactions Throughput (MRC/s)	Throughput[a]. (MCycles/s)
512	48,55	102,04	2,10	203.70
1024	98,30	204,48	2,08	201.76
2048	198,59	409,37	2,06	199.82
4096	301,08	614,39	2,04	197.88

[a] Throughput is obtained by multiplying Reactions Throughput (MRC/s) by the average number of clock cycles per reaction cycle ($L_{RC} = 97$).

B. Performance Evaluation

In order to determine the performance of our NRM SoC, to highlight the importance of the average number of dependent reactions as the critical parameter and to prove that throughput is not affected by the total number of reactions (*m*, biomodel's complexity), we simulated a Linear Chain System (LCS), which is a scalable benchmark model, having the following form:

$$S_{i \bmod m} + S_{(i+1) \bmod m} \xrightarrow{k_i} S_{(i+2) \bmod m} + S_{(i+3) \bmod m} \quad (5)$$

All reactions R_i are of 2nd order and $i = 0, 2...m-1$, with *m* in the range 512 to 4096. When a reaction channel is activated the four molecular species are affected, so the propensity of the winner reaction and another four reactions need to be updated, irrespectively of *m*. So, according to the NRM, we must re-calculate only the winners reaction time τ_μ, using equation (3), and re-normalize the times τ_α of the four dependent reactions, using equation (4).

Considering the benefits of the NRM and taking advantage of the hardware parallelization, our design is implemented in such way as to avoid the extra latency that is needed for accessing the Index Priority Queue. As discussed in Section V, when *m* increases, the height of the CoT also increases, in such a way that no extra clock cycles are needed to find the next winner reaction.

Table I summarizes the simulation results of the LCS benchmark biomodel for four different *m* values. In order to assess the performance of our implementation, we measured the total simulation time (real time - T_{real}), the total number of Reactions Cycles that occurred during the simulation and then calculated the *Reactions Throughput* (in Mega Reaction Cycles per second - MRC/sec) and the *Throughput* (in Mega Cycles per second - MCycle/sec). By keeping the operating frequency fixed (at 200 MHz) and gradually increasing the number of reactions, we show that the SoC's throughput is ∼200 MCycles/s per NRM core (called MSteps/s in [7]). The latter is only slightly affected by the biomodel's complexity (*m*) and it is practically constant even for very high *m* values, since it depends only on the average number of affected reactions. In the case of the LCS benchmark biomodels, $Daver = 4$ since

TABLE II. PERFORMANCE EVALUATION - ASYN [9] BIOMODEL

	COPASI[a]		NRM-SSA SoC	
Repetitions	8000	16000	8000[b]	16000[c]
T_{real}(sec)	1378	2768	70.58	77.6
Speed-Up	1	1	19.52	35.67
Reactions Throughput (MRC/s)	N/A	N/A	17.86	30.47
Throughput [d] (MCycles/s)	N/A	N/A	1928.88	3290.76

[a] Intel Core i7 4790K CPU at 4.4GHz with 32 GB of RAM under GNU/Linux OS.
[b] 8 NRM-SSA Cores (220MHz)
[c] 16 NRM-SSA Cores (200MHz)
[d] Throughput is obtained by multiplying Reactions Throughput (MRC/s) by the average number of clock cycles per reaction cycle $L_{RC} = 108cc$

by construction $D = d - 1 = 4$ for all reactions (see formula 5). So, if we configure our NRM SoC to have *C* NRM cores, then the throughput can be multiplied by *C*, to effectively become $C * 200$ MCycles/s. Thus, it becomes apparent that by using a large enough FPGA and a SoC with more than 5 cores it becomes possible to reach a throughput in the range of GCycles/s.

The NRM SoC was validated by simulating a curated biomodel from the EBI Biomodels database. As we discussed earlier, ASYN is a medium-size biomodel characterized by $m = 136$, $n = 90$ and $D_{aver} = 16$. As Figure 1 shows, the reactions are not affected in the same way and so the dependencies profile is not uniform as for the LCS benchmark biomodels. In this case, the PE re-normalizes a different amount of τ_α's and therefore its latency (L_{RC}) in not the same for all RCs.

Due to the fact that ASYN models a real world biological system, we should execute a large number of stochastic repetitions in order to assess the stochastic behaviour of the molecular species trajectories. To achieve high performance, we configured the NRM SoC to have more than one core. Each core is independent, using a different pseudo-random number sequence and assigned to perform 1000 repetitions of the simulation. The configuration parameters of each core are $m = n = 256$, $D_{aver} = 16$ and $H = 2$. Each repetition simulates the ASYN bio-model for $T_{sim} = 1week$ and the chosen sampling period is $T_{sam} = 1hour$. Finally, we configured COPASI to execute the same simulation.

Table II presents the results of two different configurations of the NRM SoC as compared to COPASI. The simulation results not only validate the NRM SoC's architecture but also demonstrate the efficiency of our FPGA hardware accelerator relatively to a popular software solution. Comparing the execution times of software and hardware realizations, we observe that our MPSoC with 8 NRM cores running at 220 MHz achieves a speedup factor of 19.52, while the MPSoC with 16 NRM cores at 200 MHz achieves an almost double speedup factor of 35.67. Moreover, the throughput of our MPSoC with 16 NRM cores is more than 3 GCycles/sec proving the high efficiency of our implementation and its optimal scaling characteristics when the workload of the simulation repetitions is split among independent cores.

VII. CONCLUSIONS

The demonstrated efficiency and scalability of our NRM MPSoC accelerator for stochastic simulation is a result of a flexible configurable IP core and a well designed parallel architecture for FPGAs. To maximize performance we split the simulation repetitions workload on many independent NRM cores, each one capable of simulating biomodels with up to $m = 4096$ reactions on a medium size FPGA device. To maximize throughput, the core's design minimizes the reactions cycle latency which depends only on the outdegree of the biomodel's DG and not on its complexity. The independent operation of the NRM cores in the MPSoC circumvents scalability problems that prior implementations were facing, due to their need for communication and synchronization of shared modules. By simulating real and benchmark biomodels we have demonstrated the high speedup gains of the MPSoC's FPGA implementation against a popular and efficient software simulator. Our MPSoC can reach performance in the range of Gcycles/sec, the highest reported in the literature, even when using a medium size FPGA running at ∼200 MHz.

ACKNOWLEDGEMENTS

This work has been supported by research grant "StochSoCs" No 3828 (Manolakos, PI) implemented within the framework of the operation "ARISTEIA II", co-funded by the European Union (European Social Fund) and national resources through the Operational Programme "Education and Lifelong Learning".

REFERENCES

[1] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[2] D. T. Gillespie, "Stochastic Simulation of Chemical Kinetics," *Annual Review of Physical Chemistry*, vol. 58, no. 1, pp. 35–55, 2007.

[3] M. A. Gibson and J. Bruck, "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," *The Journal of Physical Chemistry A*, vol. 104, no. 9, pp. 1876–1889, 2000.

[4] M. Yoshimi *et al.*, "An FPGA Implementation of High Throughput Stochastic Simulator for Large-Scale Biochemical Systems," in *IEEE Field Programmable Logic and Applications*, 2006, pp. 1–6.

[5] O. Hazapis and E. Manolakos, "Scalable FRM-SSA SoC Design for the Simulation of Networks with Thousands of Biochemical Reactions in Real Time," in *IEEE Field Programmable Logic and Applications (FPL)*, Sept 2011, pp. 459–463.

[6] M. Yoshimi *et al.*, "Practical implementation of a network-based stochastic biochemical simulation system on an FPGA," in *IEEE Field Programmable Logic and Applications (FPL)*, Sept 2008, pp. 663–666.

[7] D. Thomas and H. Amano, "A fully pipelined FPGA architecture for stochastic simulation of chemical systems," in *IEEE Field Programmable Logic and Applications (FPL)*, Sept 2013, pp. 1–7.

[8] S. Hoops *et al.*, "COPASI - a COmplex PAthway SImulator," *Bioinformatics*, vol. 22, no. 24, pp. 3067–3074, 2006.

[9] E. Ouzounoglou *et al.*, "In silico modeling of the effects of alpha-synuclein oligomerization on dopaminergic neuronal homeostasis," *BMC Systems Biology*, vol. 8, no. 1, p. 54, 2014. [Online]. Available: http://www.ebi.ac.uk/biomodels-main/BIOMD0000000559

[10] M. Hucka *et al.*, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[11] C. Li *et al.*, "BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models." *BMC Systems Biology*, vol. 4, p. 92, Jun 2010.

[12] E. Ouzounoglou *et al.*, "Modeling of alpha-synuclein effects on neuronal homeostasis," Model of the Month, *EMBL-EBI BioModels Database,* Available: https://www.ebi.ac.uk/biomodels-main/static-pages.do?page=ModelMonth/2015-03, [Mar, 2015]