Accelerating Epidemic Modeling through Reconfigurable Network on Chip Architectures

***Abstract*—Spreading dynamics of many real-world processes lean heavily on the topological characteristics of the underlying contact network. With the rapid temporal and spatial evolution of complex inter-connected networks, microscopic modeling and stochastic simulation of individual-based interactions have become challenging in both, time and state space. Driven by the surge to reduce the time complexity associated with system behavior analysis over different network structures, we propose a network-on-chip (NoC) based FPGA solution. The proof of concept is supported by the design, implementation and evaluation of the classical heterogeneous susceptible-infected-susceptible (SIS) epidemic model on a scalable network-on-chip (NoC) architecture on a cloud platform. The steady-state results from the proposed NoC implementation for the fractions of susceptible and infected nodes are shown to be comparable to those acquired from software simulations, but in a significantly shorter time period. Analogous to network information diffusion, implementation of the SIS model and its variants will be beneficial to foresee possible epidemic outbreaks earlier in time and expedite control decisions.**

***Index Terms*—Epidemic modeling, node-level dynamics, network on chips, reconfigurable computing, hardware acceleration**

I. Introduction

Epidemic modeling is an effective mathematical tool widely adopted in many domains to quantify the spreading dynamics of processes intertwined with large-scale networks [1], [2]. It serves as a viable framework for analyzing information diffusion in social networks [3], [4], cascading failure in power grids [5], secure routing in communication networks [6], [7], and digital virus spreading in wireless mobile networks [8], [9]. Dynamical models at the network scale can be broadly classified into two types: *deterministic* and *stochastic*. In deterministic models, the network is divided into smaller groups, each representing a specific stage of the epidemic. Such models, often formulated as differential equations (in continuous time) or difference equations (in discrete time), abstract what happens on the average at the network level. In contrast, a stochastic model is formulated as a stochastic process which in turn, is a set of random variables, $X(t;\varpi) \equiv X(t)$, defined as $\{X(t;\varpi)|t \in T \text{ and } \varpi \in \Omega\}$ where T and Ω represent time and a sample space, respectively. The solution of a stochastic model is a probability distribution for each of the random variables. Such models allow follow-up of each node in the network randomly [10], [11].

Similar to the spreading of pathogens in biological systems, the virulence of spreading processes depends not only on the infection rate of each node, but also on the connectivity of the underlying network structure [12]. Increase in computational power over recent years has revealed the existence of heterogeneous and multi-faceted relations in the description of various diffusion processes [13]. Limited mainly by memory space, the integration of such complex features in understanding the course of epidemic outbreaks using sophisticated simulation tools becomes time demanding as the network grows in size. The time taken to project the spreading pattern is important in devising effective countermeasures to prevent any potential outbreaks.

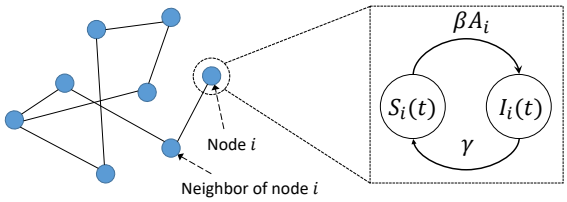


Fig. 1. Schematic of the SIS model where A_i denotes the direct neighbors of node i in the network adjacency matrix A.

We propose a network-on-chip (NoC) model supported on a reconfigurable platform to accelerate the epidemic projection of the spreading model. Due to its inherent parallelism, hardware implementation may significantly outperform pure software implementation especially due to the concurrent friendly nature of the model. The NoC model is easily scalable to support larger network sizes, only limited by the resource availability of the target FPGA devices. Recent introduction of FPGA-accelerators in the cloud environment is especially encouraging in this regard, where users can choose the target device and scale it based on their computing requirements [14]. Since the cloud-model charges users based on the target platform type and their running time, accelerated computation can significantly reduce the cost of computation. With minor modifications, the proposed platform can be used for hardware acceleration of other epidemic spreading models as well as spreading dynamics in contact networks in general.

The remainder of this paper is organized as follows. Section II discusses the background of epidemic models and related works. Section III presents the detailed architecture of the proposed hardware platform. Section IV discusses the hardware implementation results and the comparison with corresponding software implementation and Section V finally concludes the paper and gives future research directions.

II. Background and Related Works

Epidemic models are used to predict the progression of infectious diseases in a given population and the likely outcome of an epidemic. The classical susceptible-infected-susceptible (SIS) model depicted in Fig. 1 serves as the basis for many extended models, wherein $S_i(t)$ and $I_i(t)$ denote the probability of node i

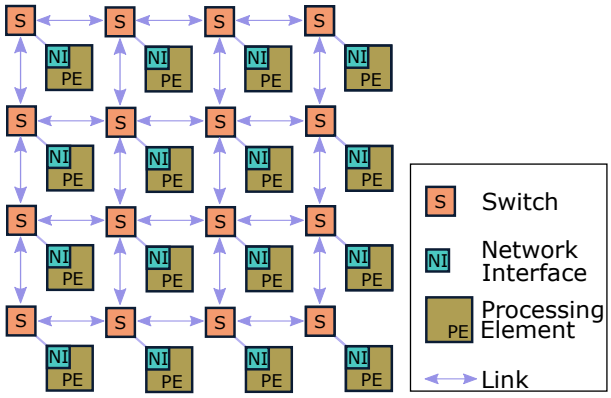


Fig. 2. Proposed NoC-based platform with mesh topology showing switch interconnections and network interfaces.

being in the susceptible (S) or the infected (I) state, repectively, in a network of size N [1]. Nodes that recover from the infection immediately transition to being susceptible again. The discrete-time node-level SIS epidemic model has the following form:

$$S_i(t) = \beta S_i(t-1) \sum_{j=1}^{N} a_{i,j} p_j(t) - \gamma p_i(t), \tag{1}$$

satisfying the condition $S_i(t) + I_i(t) = 1$ for all t values. Here, β denotes the rate at which node i gets infected, γ is the recovery rate, p_i is the probability of i being infected at time t, and a_{ij} is any element in the adjacency matrix A corresponding to the network defined as:

$$A = \{a_{ij}\} = \begin{cases} 1, & \text{if nodes } i \text{ and } j \text{ are connected neighbors} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Advancements in network science led to the revival of several unique recurring patterns inherent in networks which essentially drive the spreading pattern of processes. The Erdös-Rényi (ER) model was the first to be used for generating typical random networks [15]. An ER network of N nodes, wherein a link is included independent of other links with probability p, has a mean link count of $\binom{N}{2}p$, mean degree of $(N-1)p$, and binomial degree distribution. Such networks manifest low degree heterogeneity (most nodes have the same degree), low clustering coefficient (probability that two neighbors of a node are also neighbors), and short average path length. While ER networks are highly robust against deliberate attacks, they lack the large degree of transitivity witnessed in reality. To overcome this shortcoming, the Watts-Strogatz (WS) model was proposed to generate random graphs with small-world properties by rewiring the links of a lattice with some given probability [16]. The Barabási-Albert (BA) model was then developed to generate random scale-free networks with high degree heterogeneity. Based on the concept of preferential attachment or "the rich gets richer", the network initially begins with at least two nodes where a newly added node is most likely to connect to nodes with higher degrees. This results in the formation of a few highly-connected nodes in the network.

Except for a few, most of the existing simulation tools support deterministic modeling of simplified processes. EpiModel [17] is an R package to analyze stochastic individual and network-level epidemic models. A stochastic simulator for generalized epidemic modeling known as GEMFsim [18] has been reported and made available in MATLAB, R, Python, and C programming platforms. These simulators however, demand longer running time as the scale and complexity of the network increases.

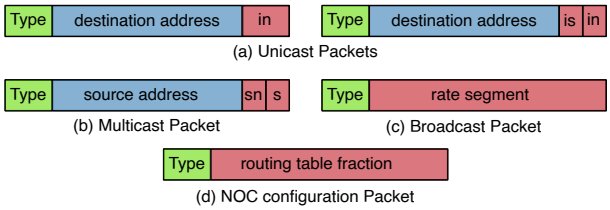


Fig. 3. NeuroNoC packet formats for supporting neuron and network configurations and data communication.

In this work, we propose to emulate the SIS process on an NoC platform. NoC is an interconnect approach that helps different subsystems in a system to communicate with each other in a scalable manner [19]. In this approach, each processing element (PE) is connected to a switch and multiple switches are interconnected to form a network. They follow packet switched communication paradigm which makes them highly scalable. In the past, NoCs have been successfully used in many applications including image and signal processing [20], neural networks [21], multi-processor systems [22], and virtual machines [23]. To the best of our knowledge, this is the first application of NoCs for accelerating epidemic models. Due to the inherent similarity in architectures, NoCs appear to be ideal candidates for mapping different network models encountered in spreading models.

III. Architecture

The proposed NoC-based platform follows mesh topology with each processing element (PE) along with its network interface (NI) representing a *node* in the contact network. Nodes are interconnected with the help of switches and bi-directional physical links as shown in Fig. 2. The NoC configuration and inter-node communication are supported via packet switching. The bottom left switch acts as the communication interface with external world, through which configuration packets are sent as well as the network status is monitored. An external host such as a server computer configures the NoC for the target network and monitors the network status as time progresses. By analyzing the packets received from the network, the host can determine the specific nodes that are infected, nodes that have recovered, and the overall spreading pattern of the process.

A. Packet Formats

The NoC manages configuration as well as inter-node communication using the different packet formats shown in Fig. 3. It supports unicast, multicast and broadcast packet transmissions based on the packet *type*. Unicast packets are used for node configuration at the beginning of the simulation (at zero epoch or at $t = 0$) by an external host. The target node address (X and Y coordinates of the node) is stored in the *destination address* field and the configuration data are carried in the *input number (in)*

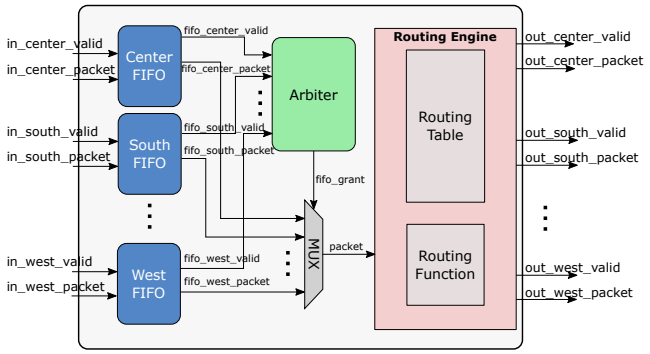


Fig. 4. The NoC switch architecture with store-and-forward functionality and support for unicast and multicast routing.

and *initial status (is)* fields. The *input number* configures the number of neighbors (number of nodes connected to this node based on the adjacency matrix) and the *initial status* configures whether the node is infected or susceptible at zero epoch.

Multicast packets are used for inter-node communication, where each node updates all its neighbors with its status after each epoch (each discrete time in simulation). Rather than sending the same packet to each of its neighbors, each node injects a single packet to the NoC and the unique router design duplicates the packets close to the target nodes. Doing so considerably reduces the network routing congestion and improves the overall latency. The packet carries the address of the injecting node in the *source address* field and the status (infected/susceptible) in the *status (s)* field. Due to the packet switched nature of the NoC, it is possible that packets are delivered out-of-order to the destination nodes. To manage this, each multicast packet carries a *sequence number (sn)* field, which specifies the discrete simulation time. Every packet in the network originating at the same discrete simulation time will have the same sequence number.

Since all nodes in the network share the same infection (β) and recovery rates (γ), this information is broadcasted across the network at the beginning of the simulation. Again, the router design and the routing algorithm enables injecting a single packet from the external host and the packet being replicated and delivered to each node. The *rate segment* of the packet initializes a portion of the pseudo random binary sequence (PRBS) generator used in the network interface discussed in Section III-C. Since the current implementation uses a 100-bit long PRBS and the rate segment is only 10-bits wide, 10 configuration packets are required to initialize them. NoC configuration packets are special broadcast packets that configure the routing tables (RTs) inside the switches. Each packet configures a portion of an RT and multiple packets are required to configure the entire network.

B. Switch

The overall architecture of the NoC switch is depicted in Fig. 4. The switch follows store and forward architecture with each interface (from 4 neighboring switches and the node) connected to an input FIFO. All the switch interfaces follow AXI4-Stream protocol [24]. To limit resource utilization, output FIFOs are omitted in the design. An arbiter chooses one of the input FIFOs for packet transmission based on their requests following a round-robin scheme. This avoids resource starvation and minimizes packet queueing delays. The *fifo_grant* signal from the arbiter drives the output of a multiplexer which selects the appropriate FIFO output for packet transmission.

The selected packet is forwarded to the routing engine (RE). The RE logic first checks for the packet type. Unicast packet (PE configuration packets) routing is managed by a routing function (RF) and multicast packet (PE status packets) routing is managed by a routing table (RT). The RF logic implements the traditional dimension-ordered XY routing by comparing the destination address embedded in the packet with the switch's address [25].

Multicast routing scheme is deployed for status packets to reduce the network congestion and latency. This method also frees nodes from storing the RTs thus, making their design relatively simple. Each entry in the RT used for multicast routing is 5 bits wide and the RT depth is same as the overall network size (number of nodes in the network). The source address embedded in the multicast packets serves as the RT entry number. Each bit in a table entry determines the directions in which a packet originating from the corresponding address will be forwarded. The broadcast could be to one or more of the neighboring switches as well as the to the node interfaced with the switch. By appropriately configuring the RTs, packets from any node can be broadcasted to any given subset of nodes within the network. The routing path taken by each packet is similar to *tree routing*, where the root of a tree is the source node and the destination nodes are located at the tree branches and leaves. If the destination nodes are located along the tree branches, intermediate switches perform forward-and-absorb operation. Traditional tree routing suffers from the possibility of deadlocks [26] in the intermediate nodes, but combining it with XY routing circumvents this possibility.

The content of each RT is determined offline by an application based on the network adjacency matrix. For a network with *NETWORK_SIZE* nodes, aspect ratios *X* and *Y*, and adjacency matrix *adjacencyMatrix[NETWORK_SIZE][NETWORK_SIZE]*, each table entry *i* corresponding to each switch *j* is generated based on Algorithm 1. The entire RT is injected to the network through the bottom-left switch as NoC configuration broadcast packets. This approach is taken to keep packets sizes small, as packets do not carry information regarding router and RT address. From the received broadcast packets, each switch selects only the portions corresponding to its RT and transmits the entire table to the neighboring switch to its right. Switches along the first column of the NoC transmits these packets to the neighboring switches in the north direction as well. Each packet carries only a fraction of the table (10-bits or 2 entries) and may require thousands of packets for complete configuration.

C. Network Interface (NI)

The NI module manages the communication between a switch and the corresponding PE. Moreover, it also implements the logic to control the state of the node after each discrete simulation time. Its detailed architecture is depicted in Fig. 5. The *working mode state machine* (WMSM) manages the operating mode of a node, which may be either in *configuration state* or in *running state*.

Algorithm 1 Routing Table Generation

```
1: Clear all RT entries.
2: for j =0; j<NETWORK_SIZE; j=j+1 do
3:     for i =0; i<NETWORK_SIZE; i=i+1 do
4:         if adjacencyMatrix[j][i]==1 then
5:             inAddr=i;
6:             outAddr=j;
7:             RT[i][j][CENTER] = 1;
8:             if outAddr [X]> inAddr [X] then
9:                 for x=outAddr[X]; x>inAddr[X];x=x-1 do
10:                    RT[outAddr[Y]×XSIZE+x][j][WEST]=1
11:                end for
12:            else
13:                for x=outAddr[X];x<inAddr[X];x=x+1 do
14:                    RT[outAddr[Y]×XSIZE+x][j][EAST]=1
15:                end for
16:            end if
17:            if outAddr[Y]>inAddr[Y] then
18:                for y=outAddr[Y];y>inAddr[Y];y=y-1 do
19:                    RT[y×XSIZE+inAddr[X]][j][SOUTH]=1
20:                end for
21:            else
22:                for y=outAddr[Y];y<inAddr[Y];y=y+1 do
23:                    RT[y×XSIZE+inAddr[X]][j][NORTH]=1
24:                end for
25:            end if
26:        end if
27:    end for
28: end for
```

When the network is in the configuration state, WMSM routes the configuration packets received from the switch interface to appropriate destination registers. The contents of the unicast packet specifying the number of neighbors of the node is stored in the *inputNum* register.

The discrete time probabilities required to decide the state of a node (whether infected or susceptible) are implemented by the *gamma* and *beta* PRBS generators. Both are linear feedback shift registers (LFSRs) composed of 100 flip-flops with the last stage feeding back to the first stage. In order to implement a specific rate (infection or recovery), the corresponding probability is multiplied by 100 and the LFSR is initialized with a random binary pattern with number of ones equal to the result of multiplication. For example, to achieve a β value of 0.3, the 100 flip-flops in the *beta* LFSR are initialized with a random binary pattern with 30 ones and 70 zeros. The initialization values for the two PRBS generators are received as broadcast packets from the external host. Since the size of PRBS generators is much larger than the packet size, multiple configuration packets are required to initialize them. The *Input Counter (IC)* logic specifies the index number of the PRBS generators to which the incoming configuration packet values are written.

After initialization, the LSFRs freely run and the least significant bit (LSB) is used for achieving the required rate. Due to the initialization pattern, the probability of the LSB becoming one will be same as the required rate. This is similar to finding a number between 0 and 1 from a uniform distribution and checking whether its value is less than the required rate. The initialization patterns are generated offline through a software application and broadcasted to the nodes through multiple packets as discussed

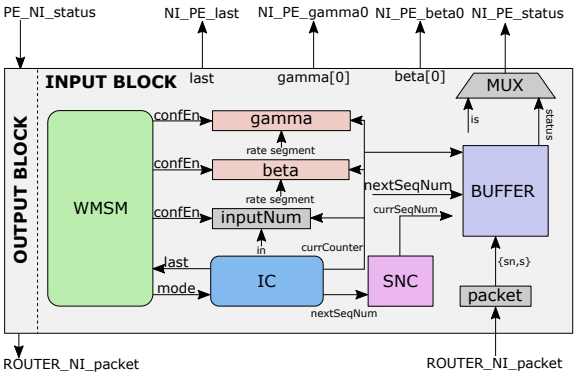


Fig. 5. Network Interface Architecture.

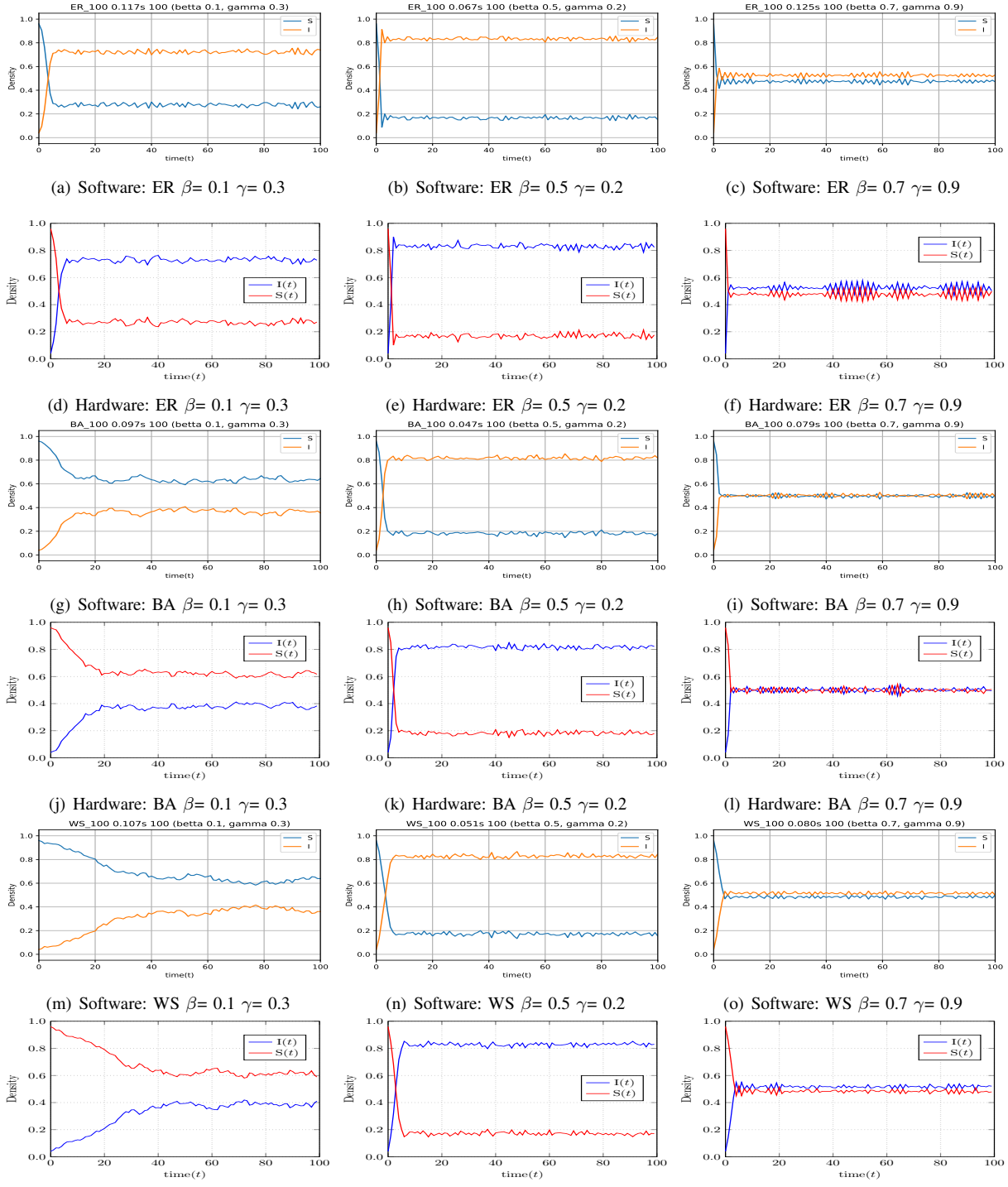
before. Although same initialization packets are broadcasted to all the nodes, due to the inherent latency in packet switching, each LFSR will have a different initialization pattern at epoch zero but representing the same rate.

Once the WMSM receives the unicast packet carrying the initial node status, it is transferred to the PE and is switched to *running state*. When a PE receives its initial status, it is immediately broadcasted to its neighbors. Thus,some PEs may possibly receive packets from its neighbors before the end of the configuration. Moreover, due to path length differences and different congestion levels along different paths, out-of-order packets may arrive even at running state which may result in a node receiving a status from one of its neighbors for the next discrete time before receiving all status for the current time.

To overcome the out-of-order status delivery, PEs embed a sequence number into the status packets. The sequence number indirectly represents the current discrete time. When status packets are received, they are initially stored in a buffer with logical partition for each sequence number. Each partition can store NETWORK_SIZE number of status and each entry is just one 1 bit wide (to represent infected or susceptible). Each partition maintains its own counter which counts the number of status that have already arrived and to which entry the next status will be updated. In running mode, the status are extracted from a partition specified by the *sequence number checker* (SNC) one at a time and transferred to the PE. The status are sent to the PE only after receiving the status from all its neighbors, whose number is specified in the *inputNum* register during configuration. Once all status are sent, the SNC is incremented to select the next partition. Since each PE generates a status packet only after receiving the status from all its neighbors, it could be proven that two sequence numbers are enough to distinguish between different discrete times. Thus, the size of the buffer is 2×NETWORK_SIZE bits to support two partitions.

D. Processing Element (PE)

Each PE runs the SIS state-machine, similar to the one shown in Fig. 1. The initial infection status is received from the NI during the configuration stage. Once the NI receives status from all the neighbors for a discrete time, it transfers the status one at a time to the PE. If the PE is in susceptible state and receives an infected status from one of the neighbors, it checks the output of



(a) Software: ER β= 0.1 γ= 0.3 (b) Software: ER β= 0.5 γ= 0.2 (c) Software: ER β= 0.7 γ= 0.9

(d) Hardware: ER β= 0.1 γ= 0.3 (e) Hardware: ER β= 0.5 γ= 0.2 (f) Hardware: ER β= 0.7 γ= 0.9

(g) Software: BA β= 0.1 γ= 0.3 (h) Software: BA β= 0.5 γ= 0.2 (i) Software: BA β= 0.7 γ= 0.9

(j) Hardware: BA β= 0.1 γ= 0.3 (k) Hardware: BA β= 0.5 γ= 0.2 (l) Hardware: BA β= 0.7 γ= 0.9

(m) Software: WS β= 0.1 γ= 0.3 (n) Software: WS β= 0.5 γ= 0.2 (o) Software: WS β= 0.7 γ= 0.9

(p) Hardware: WS β= 0.1 γ= 0.3 (q) Hardware: WS β= 0.5 γ= 0.2 (r) Hardware: WS β= 0.7 γ= 0.9

Fig. 6. Comparison of software simulation and hardware simulation outputs for different β and γ values in a 10×10 network.

the *beta* PRBS generator. If the PRBS output is high, the PE state is switched to *infected*. Further status received from the neighbors are ignored for the current epoch since, in the SIS model, a node cannot change its state more than once in any discrete time. If the PE is in *infected* state, it checks the output from *gamma* PRBS generator after receiving status from all neighbors. If the PRBS output is high, the state is switched to *susceptible* and the status of the neighbors play no role in this state transition. In all cases, the status of the PE after a discrete time is sent to the NI module which is then multicasted to all its neighbors.

The SIS state-machine is relatively simple, but due to the modular design approach any other model can be easily incorporated by modifying this state-machine alone. In future, as cloud FPGA instances start supporting partial reconfiguration (PR), models can be dynamically updated by only reconfiguring the PE module which substantially reduces the design and configuration time.

TABLE I
RESOURCE UTILIZATION OF THE PROPOSED ARCHITECTURE FOR 16×16 IMPLEMENTATION WHEN TARGETING XILINX ULTRASCALE+ VU9P

Module Name	Slice LUTs	Slice Regs	Memory Block
Network Interface	322	251	0
Processing Element	7	7	0
Switch	299	351	0
Total per node	628	609	0
Total Network	**159710**	**156290**	**0**

E. Simulation Steps

The host computer executes the following steps for the proposed NoC-based SIS model simulation:

- Inject the NoC configuration broadcast packets to configure the routing tables (Fig. 3d).
- Inject the broadcast packets to configure β and γ (Fig. 3c).
- Inject the unicast packets to configure the number of neighbors to each node (Fig. 3a).
- Inject the unicast packets to configure the initial status of each node (Fig. 3a).
- Receive packets from the NoC and monitor the network status. Once status packets are received from all nodes, increment the discrete time step and log the number of infected and susceptible nodes.

IV. RESULTS AND DISCUSSION

The proposed platform is designed with Verilog HDL and implemented on a Xilinx Ultrascale+ VU9P device targeting Amazon AWS cloud-based FPGA instances. Module-wise utilization for important building blocks and the overall utilization for a 16×16 network (256 nodes) is given in Table I. The implementation consumes about 8% LUTs and 3% of flip-flops on the target device. At this rate, a network of 3000 nodes can be supported on this device after reserving enough resources for the communication infrastructure (AWS shell infrastructure). FPGA architecture-dependent resources such as BRAM/URAM tiles of DSP slices are not utilized making the design highly portable to other platforms such as the Microsoft Azure or on-premises implementations. Due to the heavily pipe-lined implementation, the platform can support to up to 260MHz clock frequency, but is restricted to 250MHz due to the PCIe-based host system interface.

We first verified the validity and the functional correctness of the proposed platform by comparing its output with corresponding Matlab software-based implementation of the discrete SIS model. Tests were conducted for three different network sizes (10×10, 16×16, and 32×32) for three β and γ values and three different topologies, thus giving a total of 27 test cases. Results from each test case were averaged over 10 runs to avoid outliers. Software and hardware test outputs corresponding to a 10×10 implementation (network with 100 nodes) with different rate and topology configurations shown in Fig. 6. It reveals that the steady state behavior and the number of discrete steps required to reach the steady state are similar in both cases which validates the functional correctness of the platform.

TABLE II
WALL CLOCK TIME REQUIRED FOR SIMULATING 100 DISCRETE TIME STEPS IN SOFTWARE AND PROPOSED IMPLEMENTATION

Topology	**Run time(ms)**	**Network Size**		
		100	**256**	**1024**
WS	Software	107	312	1297
	NoC	0.213	0.232	2.81
BA	Software	97	242	1130
	NoC	0.215	0.248	2.78
ER	Software	117	473	4408
	NoC	0.217	0.262	3.67

Table II compares the total run-time of software and the corresponding NoC-based implementations for modeling 100 discrete time steps for different network models and sizes. The software runs on an *AWS EC2 a1.xlarge* cloud compute instance with 4 vCPUs and 8 GB RAM. The NoC runs on an *AWS f1.2xlarge FPGA* at 200 MHz clock frequency. The FPGA run-time includes the time required for configuring the RT each time before starting the simulation. In a practical scenario, this could be avoided as long as the network topology remains intact. The run-time for 10×10 and 16×16 are very similar for NoC implementation since the 10×10 implementation is physically a 16×16 implementation mapped using appropriate adjacency matrix. This also shows the flexibility of the NoC architecture where a sub-network with any topology can be mapped to a mesh architecture using appropriate adjacency matrix. It is evident from the data that hardware outperforms software by an order of 2 to 3. It might be possible to improve the software performance by implementing the model in native C language instead of compiled MATLAB. Yet the hardware performance will be better by at least by an order. Considering the hourly rate of 0.102 USD for a1.xlarge instance and $0.76 - 1.65$ USD for (based on subscription type) for f1.2xlarge instance, hardware acceleration can provide considerable financial benefits to users.

V. CONCLUSION

In this paper, we discussed the design, implementation and performance evaluation of a network on chip based solution for epidemic modeling. Experimental results show that the proposed architecture can accelerate the spreading model by many orders compared to software implementation on cloud-infrastructure and can provide significant financial benefits to users. The proposed model can be easily adapted to other scenarios such as social information diffusion, wireless malware propagation and viral marketing. It is shown that a hardware-based solution can significantly outperform software simulation in terms of run-time and at the same time, provide scalability due to the NoC-based architecture. As future work, we intend to extend and demonstrate the effectiveness of the platform in modeling multiple competitive processes in multi-layered networks. We believe that modeling of spreading dynamics will provide new research directions to the hardware accelerator research community. The HDL implementation and the dataset used for evaluation are available as open source from the following git repository [27].

REFERENCES

[1] E. Vynnycky and R. White, *An Introduction to Infectious Disease Modeling*. Oxford University Press, 2010.

[2] M. Keeling and P. Rohani, *Modeling Infectious Diseases in Humans and Animals*. Princeton University Press, 2011.

[3] A. Dadlani, M. S. Kumar, M. G. Maddi, and K. Kim, "Mean-field dynamics of inter-switching memes competing over multiplex social networks," *IEEE Communications Letters*, vol. 21, no. 5, pp. 967–970, 2017.

[4] Y. Chen, Y. Mao, L. Cui, S. Leng, Y. Wei, and X. Chen, "A two layer model of malware propagation in a search engine context," in *Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 21–26.

[5] M. Korkali, J. Veneman, B. Brian, F. Tivnan, J. Bagrow, and P. Hines, "Reducing cascading failure risk by increasing infrastructure network interdependence," *Nature Scientific Reports*, vol. 7, p. 44499, 2017.

[6] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," Duke University, Tech. Rep., Apr. 2000.

[7] S. Cheng, P. Chen, C. Lin, and H. Hsiao, "Traffic-aware patching for cyber security in mobile iot," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 29–35, 2017.

[8] S. Peng, S. Yu, and A. Yang, "Smartphone malware and its propagation modeling: A survey," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 925–941, 2014.

[9] L. Yang, X. Yang, and Y. Y. Tang, "A bi-virus competing spreading model with generic infection rates," *IEEE Transactions on Network Science and Engineering*, vol. 5, no. 1, pp. 2–13, 2018.

[10] F. Brauer and C. Catillo-Chavez, *Mathematical Models in Population Biology and Epidemiology*. Springer-Verlag, 2001.

[11] T. Britton, "Stochastic epidemic models: a survey," *Mathematical Biosciences*, vol. 225, no. 1, pp. 24–35, 2010.

[12] P. Van Mieghem, *Performance Analysis of Complex Networks and Systems*. Cambridge University Press, 2014.

[13] M. Newman, *Networks: an Introduction*. Cambridge University Press, 2014.

[14] P. Patel. (2018) FPGA-based accelerated cloud computing with AWS EC2 F1 and SDAccel. Xilinx Inc. [Online]. Available: http://www.isfpga.org/fpga2018/slides/

[25] S. D. Chawade, M. A. Gaikwad, and R. M. Patrikar, "Review of XY routing algorithm for network-on-chip architecture," *International Journal of Computer Appllications*, vol. 43, pp. 20–23, 2012.

[15] P. Erdős and A. Rényi, "On the evolution of random graphs," in *Publication of the mathematical institute of the Hungarian academy of sciences*, 1960, pp. 17–61.

[16] A. Barabási and M. PÃ3sfai, *Network Science*. Cambridge University Press, 2016.

[17] S. M. Jenness, S. M. Goodreau, and M. Morris, "Epimodel: An R package for mathematical modeling of infectious disease over networks," *Journal of statistical software*, vol. 84, no. 29731699, p. 8, Apr. 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/PMC5931789/

[18] F. D. Sahneh, A. Vajdi, H. Shakeri, F. Fan, and C. Scoglio, "Gemfsim: A stochastic simulator for the generalized epidemic modeling framework," *Journal of Computational Science*, vol. 22, pp. 36 – 44, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877750317305227

[19] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[20] J. Joshi, K. Karandikar, S. Bade, M. Bodke, R. Adyanthaya, and B. Ahirwal, "Multi-core image processing system using network on chip interconnect," in *Proceedings of Midwest Symposium on Circuits and Systems*, Aug. 2007, pp. 1257–1260.

[21] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, Dec 2013.

[22] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. D. Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, Feb 2005.

[23] G. Mathias and K. Kent, "An embedded Java virtual machine using network-on-chip design," in *Proceedings of IEEE International Workshop on Rapid System Prototyping*, 2006.

[24] *UG761: AXI Reference Guide*, Xilinx Inc., Mar. 2011.

[26] F. A. Samman, T. Hollstein, and M. Glesner, "Adaptive and deadlock-free tree-based multicast routing for networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1067–1080, July 2010.

[27] Blanked for Review. (2019) SiSNoC Git repository. [Online]. Available: https://github.com/dsdnu/sisNoC