To the Graduate Council:

I am submitting herewith a thesis written by Brandon Parks Thurmon entitled "Reconfigurable Hardware Acceleration of Exact Stochastic Simulation." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

_____
Dr. Gregory D. Peterson, Major Professor

We have read this thesis
and recommend its acceptance:

_____
Dr. Donald W. Bouldin

_____
Dr. Chris D. Cox

Accepted for the Council:

_____
Vice Chancellor and
Dean of Graduate Studies

Reconfigurable Hardware Acceleration of Exact Stochastic Simulation

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee

Brandon Parks Thurmon
August 2005

DEDICATION

This thesis is dedicated to my devoted wife, Saneta, for her enduring patience, steadfast encouragement, and soothing compassion; our parents and family for their guidance and reassurance; and my brother, Michael, for his support and sense of humor.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all of those who have contributed their expertise and directed me towards the completion of my Master of Science degree in Electrical Engineering. First and foremost, I wish to thank my advisor, Dr. Greg Peterson, for his wisdom and insight. I would like to thank Dr. Don Bouldin for empowering me with an understanding of microelectronic design and for serving on my committee. I would also like to thank Dr. Chris Cox for serving on my committee and sharing his knowledge with me. In addition, I would like to express my thanks to my office mate and friend, James McCollum, for his continuous help.

I also wish to thank Dr. Michael Simpson and the National Academy of Sciences, whom funded this work under the Keck Futures Initiative.

ABSTRACT

This thesis explores the use of reconfigurable hardware in modeling chemical species reacting in a spatially homogeneous environment. The time evolution of biochemical models is often evaluated using a deterministic approach that uses differential equations to describe the chemical interactions of the model. However, such an approach treats species as continuous valued concentrations, is inaccurate for small species populations, and neglects the stochastic nature of biochemical systems. The Stochastic Simulation Algorithm (SSA) developed by Gillespie is able to properly account for these inherent noise fluctuations. This allows the SSA to accurately project the time evolution of a biochemical model. Unfortunately, the SSA can be computationally intensive and require a substantial amount of time to complete. Therefore, it has been proposed that the SSA be implemented on a Field Programmable Gate Array (FPGA) to improve performance. Employing an FPGA allows parallelism to be exploited within the SSA providing a speedup over software implementations executing instructions sequentially. Recent work in this area has focused on implementing the SSA on an FPGA to simulate specific biochemical models. However, this requires re-constructing and re-synthesizing the design in order to simulate a new biochemical system. This work examines the use of a reconfigurable computing platform to allow an implementation of the SSA on an FPGA to simulate a variety of models.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The future of biochemical systems analysis is as promising as it is challenging. Accurately modeling complex biochemical systems is currently a daunting and time consuming task. Efforts are underway to develop more efficient tools for modeling these systems while producing reliable data. Some biochemical systems of interest include the transcription and translation of DNA during protein synthesis or the growth of a bacterial infection, as well as many others. By understanding how cells operate and communicate, we can begin to predict the behavior of the underlying biochemical system. Then methods can be developed to interrupt and control cellular processes, allowing advances in the field of gene therapy and medicine.

Biochemical systems, consisting of species reacting in a spatially homogeneous environment, are often formulated using a deterministic approach. Such an approach represents species as continuous-valued concentrations and interactions between chemicals are modeled using ordinary differential equations. A deterministic approach is effective for modeling many biological systems, although inaccuracies become apparent for systems with small populations of chemical species and systems affected by noise. Recent research has shown that noise may play a critical role in many biochemical systems [1,2]. Therefore a stochastic approach must be used to model noise-affected systems. Within a stochastic approach, chemical species are represented as discrete-valued populations and interactions between chemicals are represented as random processes.

The Chemical Master Equation (CME) is used to define the stochastic properties of a biochemical system. The CME is typically an infinite set of differential equations, making it impractical to solve analytically for most complex systems. The Stochastic Simulation Algorithm (SSA), developed by Daniel Gillespie, is mathematically equivalent to the behavior of the CME [3].

Gillespie's algorithm simulates the execution of one chemical reaction at a time, and each simulation is a single sample of the model's behavior. In order to obtain statistically accurate results, the SSA must be executed several times to form a complete picture of the model's behavior. As a result, the SSA can be computationally intensive and time consuming, limiting its application to large-scale and biologically relevant models. Endy and Brent have suggested that a stochastic simulation of the cell cycle of a single *Escherichia coli* cell may require 100 years of computation time on today's standard PC [4].

To address these issues, this work presents a hardware-accelerated version of the SSA implemented on a Field Programmable Gate Array (FPGA). By performing tasks in parallel that would normally be handled sequentially on a regular microprocessor, the workload is divided among several process modules and the overall performance of the SSA is improved. Previous work in this area has yielded hardware simulators with impressive performance gains over software implemented simulators. However, these performance gains come at a cost. Previous designs from other researchers have focused on specific biochemical models, requiring varying levels of redesign when modeling different biochemical models. In addition, some have introduced approximations to the SSA. This work focuses on a hardware-accelerated simulator that is general purpose,

meaning several biochemical models can be simulated without the need to re-synthesis the design. Furthermore, the hardware designs presented herein remain statistically true to Gillespie's SSA. Within this work, two approaches to a general-purpose hardware implementation of the SSA are offered. The second chapter will provide a brief overview of the scope of the work. This will include an introduction to biochemical systems and how they are modeled. The third chapter will describe some of the previous work concerning hardware accelerated stochastic simulators. Chapter four will delve into the specifics of one design of a general-purpose hardware accelerated exact stochastic simulator. Chapter five will outline the details of a second design. The final chapter will present some plausible avenues for future work, in addition to conclusions from this work.

Chapter 2

Background

2.1 Applying Ordinary Differential Equations to the Model

Traditionally, models of chemical species reacting within a spatially homogeneous environment are devised using a deterministic approach involving ordinary differential equations. Such an approach treats species populations as continuous valued concentrations that are a function of time [3]. Through the use of software packages that include differential equation solvers (i.e. Matlab), a complex biological system can be modeled using ordinary differential equations (ODEs) and solved in less than a day. However, the results may not necessarily be accurate. Ordinary differential equation models ignore the inherent stochastic nature of chemically reacting systems. This hinders the application of ODEs to systems with small numbers of molecules. In addition, it is possible for the results of an ODE model to suggest that species concentrations are real valued or below zero. In actual chemical systems, it makes no sense to have any less than a whole molecule and it is impossible to have a negative amount of molecules. The effects of these limitations can be devastating to modeling chemical systems since a species with a small population can have a significant impact on the trajectory of the system.

2.2 Exact Stochastic Simulation

The Exact Stochastic Simulation Algorithm (SSA) was developed by Daniel T. Gillespie in the late 70's as a way to accurately simulate chemically reacting systems [3,7]. The SSA treats species populations as discrete values and properly handles the

randomness and noise inherent in many chemically reacting systems. In addition, the SSA exhibits the stochastic behavior evident in the time evolution of biochemical systems. Gillespie formulated two methods to perform exact stochastic simulations, the *First Reaction Method* and the *Direct Method*. Gibson and Bruck improved upon Gillespie's First Reaction Method in 2000 to develop the *Next Reaction Method* [8]. The *Optimized Direct Method* developed in 2004 by Cao, Li, and Petzold further improved the performance of exact stochastic simulations [9]. The *Sorting Direct Method*, recently developed by James McCollum, further optimized stochastic simulations [20]. A paper outlining the Sorting Direct Method was recently accepted for publication in the Journal of Computational Biology and Chemistry. All of the above algorithms simulate a possible time evolution of a chemically reacting system, determining a time for the occurrence of each reaction. Each algorithm accomplishes this through the following steps,

1: Initialization – An input model is read by the simulator and data structures are initialized.
2: Propensity Calculation – Where necessary, the propensity of each reaction is calculated based on the reaction rate constant and the current species populations.
3: Putative Time Estimation – Using the propensities and exponentially distributed random numbers, the time at which the next reaction will occur is determined.
4: Reaction Selection – The next reaction to execute is selected.
5: Reaction Execution – The species populations and system time are updated according to the execution of the selected reaction.

6: Termination – The program ends if the desired end time of the simulation has been reached. Otherwise, the process returns to the Propensity Calculation step and continues executing.

2.2.1 Explanation of Terms

The following terms are crucial to formulating an exact stochastic simulation algorithm and may require an explanation.

Propensity, *a*, is associated with the probability that a reaction will occur. It is based upon the stochastic reaction rate constant and the number of distinct molecular combinations of the reaction. The stochastic reaction rate constant, *c*, is defined as the average probability that a molecular combination from a given reaction will collide and react in the next infinitesimal time interval. The stochastic reaction rate constant is directly related to the deterministic reaction rate constant, *k*. This relationship is altered for the case when identical reactant molecules collide and react. The equation below depicts the correlation between the stochastic and deterministic reaction rate constants where n is the number of identical reactant molecules reacting together [7].

$$k_\mu = \frac{c_\mu}{n!} \qquad (1)$$

The number of distinct molecular combinations of a reaction depends on the type of reaction and the number of molecules, *X*, of each reactant of a given reaction. Table

2.1 shows the equations to some common reaction types along with the equations to calculate their propensities.

Table 2.1 – Propensity Equations

Reaction Equation	**Propensity Equation**
$A \rightarrow B, k_1$	$a_0 = X_A * k_1$
$A + B \rightarrow C, k_2$	$a_1 = X_A * X_B * k_2$
$2A \rightarrow B, k_3$	$a_2 = X_A * (X_A - 1) / 2$

Putative time, τ, refers to the amount of time it will take before a reaction occurs. The following will summarize the sampling of an exponential distribution with parameter a_i in order to determine the putative time. A uniformly distributed random number is scaled to fit an exponential distribution to find an exponential random number. The exponential random number is then divided by a propensity value to find the putative time. The following equation shows the calculation of putative time.

$$\tau_i = -\frac{1}{a_i}\log(URN) \qquad (2)$$

Other terms, specific to a certain algorithm, will be clarified as needed.

2.2.2 Gillespie's First Reaction Method

The First Reaction Method was Daniel Gillespie's first take on the SSA [3]. The Initialization step of this algorithm creates and loads variables to hold the species populations, reaction equations, and the current time. Upon the initialization of the system, the Propensity Calculation step begins and the propensity of each reaction is calculated. For each reaction during the Putative Time Estimation step, a potential time is calculated to determine when that reaction will occur in the future. Each potential time is found by generating an exponential random number and dividing it by the propensity of the reaction. The Reaction Selection step searches the list of putative times from each reaction; the reaction with the earliest time is labeled as the next reaction. The Reaction Execution step adds the putative time of the selected reaction to the current time and updates the species populations by decrementing the values of the reactant populations and incrementing the values of the product populations. This process is repeated until the desired end time is reached. See figure 2.1 to find pseudo code for Gillespie's First Reaction Method. Gillespie's First Reaction Method is an effective way to accurately model biochemical systems. However generating an exponential random number for each reaction during each iteration severely limits the method's performance.

1. Establish a list of n chemical species with their initial populations $X_1, X_2, \ldots, X_n$.
2. Establish a list of m chemical reactions and their associated stochastic rate constants $k_1, k_2, \ldots, k_m$.
3. Initialize the current time $t \leftarrow 0$.
4. Calculate the propensity $a_1, a_2, \ldots, a_m$, for each of the m chemical reactions.
5. For each reaction i, generate a putative time τ_i, according to an exponential distribution with parameter a_i.
6. Let μ be the reaction whose putative time is least.
7. Change the species populations $X_1, X_2, \ldots, X_n$, to reflect the execution of reaction μ.
8. Set $t \leftarrow t + \tau_\mu$.
9. Return to Step 4.

Figure 2.1 – Pseudo Code for Gillespie's First Reaction Method [7]

2.2.3 Gillespie's Direct Method

Gillespie formulated the Direct Method to improve the performance of the SSA [7]. The Initialization step of the Direct Method remains the same as in the First Reaction Method. The Propensity Calculation step remains the same except for the requirement that all reaction propensities be summed together. The Putative Time Estimation step is modified to find one potential time for when the next reaction will occur by generating one exponential random number and dividing by the total propensity of the system. The Reaction Selection step generates a uniformly distributed number and multiplies it by the total propensity. Then a linear search of the reaction propensities is performed, once the cumulative total of the evaluated propensities exceeds the product then the current reaction is set to be the next reaction executed. The Reaction Execution step is also the same as in the First Reaction Method. This process is repeated until the desired end time is reached. The Direct Method is able to significantly improve the performance of the SSA by requiring the generation of only one exponential random number and one uniform random number per iteration, regardless of the size of the system. See figure 2.2 for pseudo code of Gillespie's Direct Method.

1. Establish a list of n chemical species with their initial populations $X_1, X_2, \ldots, X_n$.
2. Establish a list of m chemical reactions and their associated stochastic rate constants $k_1, k_2, \ldots, k_m$.
3. Initialize the current time $t \leftarrow 0$.
4. Calculate the propensity $a_1, a_2, \ldots, a_m$, for each of the m chemical reactions.
5. Sum the propensity values: $a_{total} = \sum_{i=1}^{m} a_i$.
6. Generate a putative time, τ_μ, for the chemical system according to an exponential distribution with parameter a_{total}.
7. Choose a reaction μ using a uniformly distributed random number and a distribution of the form

   $$\Pr(\text{Reaction} = \mu) = \frac{a_\mu}{a_{total}}$$

8. Change the species populations $X_1, X_2, \ldots, X_n$, to reflect the execution of reaction μ.
9. Set $t \leftarrow t + \tau_\mu$.
10. Return to Step 4.

Figure 2.2 – Pseudo Code for Gillespie's Direct Method [7]

2.2.4 Gibson and Bruck's Next Reaction Method

Michael Gibson and Jehoshua Bruck recognized that the exact stochastic simulation algorithms originally proposed by Gillespie did not scale well to large systems with many reactions [8]. In an effort to create a more efficient SSA for exactly simulating chemical reactions, they devised the Next Reaction Method by enhancing the efficiency of the First Reaction Method. The execution time of the First Reaction Method is hindered by the following three activities that are performed with every iteration and take time proportional to the number of reactions: (1) the propensity of each reaction must be calculated, (2) a putative time must be found for each reaction, and (3) the smallest putative time must be found. The Next Reaction Method addresses each of these respective drawbacks by introducing a *Dependency Graph*, reusing putative time values, and utilizing an indexed priority queue. The dependency graph is a data structure that chronicles which reaction propensities will be affected by the execution of a given reaction. Therefore, the fewest possible number of reaction propensities are recalculated. Recall from section 2.2.2, a reaction's putative time is related to its propensity. This suggests that it is not necessary to update the putative time of a reaction whose propensity does not change. Gibson and Bruck state that typical models contain loosely coupled reactions and require only a few propensities to be updated with each time step. They make this claim to justify the use of an indexed priority queue to find the minimum putative time and subsequently the next reaction to execute. Figure 2.3 shows pseudo code for the Next Reaction Method.

1. Initialize:
   a) Set initial species populations, set $t \leftarrow 0$, generate a dependency graph G.
   b) Calculate the propensity $a_1, a_2, \ldots, a_m$, for each of the m chemical reactions.
   c) Generate a putative time, τ_i, for each reaction.
   d) Store the putative times in an indexed priority queue P.
2. Let μ be the reaction whose putative time stored in P is least.
3. Let τ be τ_μ.
4. Change the species populations $X_1, X_2, \ldots, X_n$, to reflect the execution of reaction μ. Set $t \leftarrow \tau$.
5. For each edge (μ, α) in the dependency graph G,
   a) Update a_α.
   b) If $\alpha \neq \mu$, set $\tau_\alpha \leftarrow (a_{\alpha,old} / a_{\alpha,new})(\tau_\alpha - t) + t$.
   c) If $\alpha \neq \mu$, generate a random number, ρ, according to an exponential distribution with parameter a_μ, and set $\tau_\alpha \leftarrow \rho + t$.
   d) Replace the old in τ_α value in P with the new value.
6. Return to Step 2.

Figure 2.3 – Pseudo Code for Gibson and Bruck's Next Reaction Method [8]

Gibson and Bruck also suggest applying the techniques of the Next Reaction Method to the Direct Method. In addition to including a dependency graph to update the minimal number of variables, they propose using a complete tree data structure to efficiently find the total propensity and search for the next reaction to execute. Although the details of such an algorithm are laid out, Gibson and Bruck chose not to submit a formal evaluation of such an algorithm.

2.2.5 Cao, Li, and Petzold's Optimized Direct Method

Yang Cao, Hong Li, and Linda Petzold addressed the widely held conception that the Next Reaction Method was more efficient than the Direct Method when dealing with large systems. They developed the Optimized Direct Method to outperform the Next Reaction Method [9]. They begin with a comparison of the results from the two competing algorithms when simulating several actual biochemical models. They observed that the Next Reaction Method has an advantage over the Direct Method when the system is large with loosely coupled reactions, that is to say the execution of one reaction does not affect the propensity of many other reactions. However, they determined that this is not always the case for practical problems. They also concluded that much of the Next Reaction Method's time was spent maintaining the indexed priority queue in order to determine the next reaction. After an evaluation of the previous stochastic simulation algorithms, they set out to optimize the Direct Method. They realized that in a large system some reactions are executed more frequently than others. For example, when simulating a heat shock response model that describes how E. Coli responds to a temperature increase, they found the six most frequent reactions accounted

for 95% of all executed reactions [9]. When determining the next reaction in the original Direct Method, reaction propensities are compared sequentially based upon the reaction's index. This means a reaction's index plays an important role in the search depth for the next reaction. Their group devised a way to perform a few pre-simulations on a system to determine the most frequent reactions. The reactions are then rearranged in decreasing order based on how often they execute. This optimizes the average search depth required to find the next reaction. They also appreciated the efficiency provided by a dependency graph. By applying the idea of a dependency graph, developed by Gibson and Bruck [8], only propensities of reactions affected by another reaction's execution must be recalculated. When applied to the Direct Method, subtracting the old propensities and adding the new propensities of the affected reactions can determine the sum of the propensities. Applying search depth reduction and inclusion of a dependency graph, when appropriate, made the Optimized Direct Method much more efficient than the original Direct Method.

2.3 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a semi-custom Application Specific Integrated Circuit (ASIC) that is user programmable [13]. FPGAs are prefabricated to consist of rows of logic blocks and programmable connection switches to specify interconnections. Testing a design is simplified on an FPGA, since it can be electronically programmed, erased, and then reprogrammed in a short amount of time. This is also the basis for using FPGAs in reconfigurable computing. A Hardware Description Language (HDL) or a schematic is used to define the desired functionality of

an FPGA. Typically, it is common to use an HDL to describe a large or complex design. In addition, an HDL design can be targeted to multiple layouts (including FPGAs and ASICs). The two most popular HDLs are VHDL and Verilog. An HDL allows the user to define the timing constraints and concurrency within a design. In order to prepare a design for an FPGA, the desired functionality is split into necessary blocks. Each block represents some task used towards the overall functionality. Each block is defined as an "entity" and the logic function of it is described in an HDL by a "process" that runs continuously. It is also possible for an entity to declare a "component" of another entity in order to accomplish a task. By declaring multiple processes on an FPGA, parallelism can be exploited within a design. Processes executing simultaneously can streamline a design and offer a speedup over the same design implemented in software.

2.4 Pilchard Reconfigurable Computing Platform

The Pilchard Reconfigurable Computing Platform [5] was developed to interface an FPGA to a host computer. The Computer Science and Engineering Department at the Chinese University of Hong Kong designed the Pilchard platform. Previous systems that combined the capabilities of an FPGA with a host computer utilized the Peripheral Component Interconnect (PCI) bus to handle communication between the two. The Pilchard platform uses a Dual In-line Memory Module (DIMM) slot of the host computer to interface with the FPGA. Since the memory bus is faster than the PCI bus, the Pilchard platform is able to outperform comparable systems. The host computer and FPGA are able to communicate at a maximum frequency of 133 MHz with sixty-four bit data. This provides a maximum bandwidth of 1,064 MB/s. The Pilchard platform uses a

Xilinx Virtex XCV1000E FPGA with approximately one-million gate capacity. The Virtex 1000E also contains 49,152 bytes of Block RAM [14]. The processor within the host computer is a Pentium III with a 933 MHz clock speed. The time penalty incurred when loading a design onto the FPGA is only a few seconds and is design independent. Figure 2.4 shows the circuit board of the Pilchard platform.

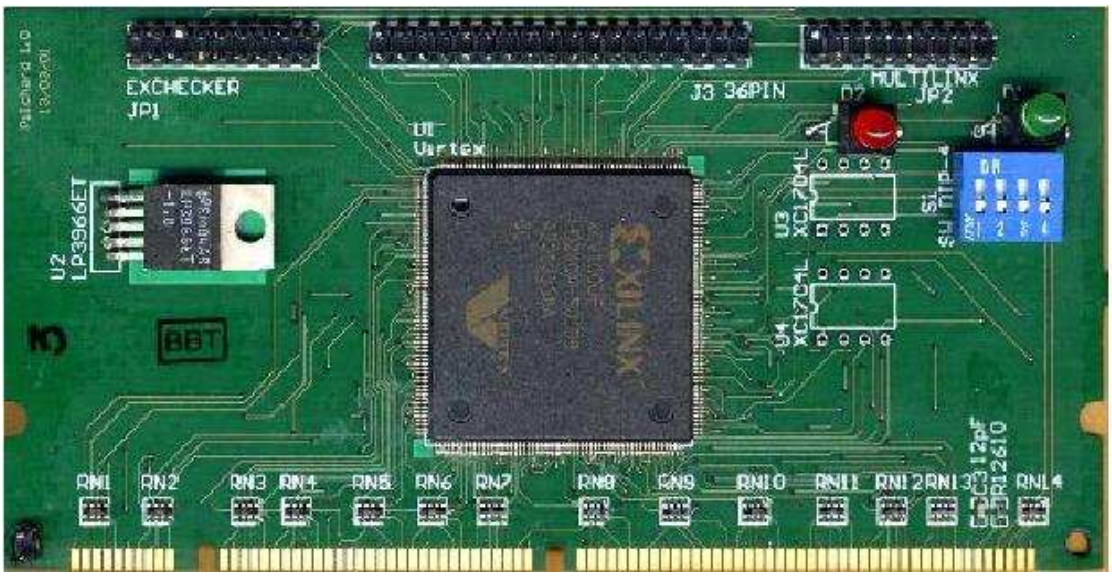

Figure 2.4 – Pilchard Platform [5]

2.5 Computing Platform

The same computing platform was used to compare the performance of the hardware implementation of the SSA against software implementations of various SSA methods. The computer that hosts the Pilchard Reconfigurable Computing Platform was also used to execute the software versions of the stochastic simulation algorithms. The computer used a Pentium III operating at 933 MHz with 256 MB of Random Access Memory (RAM). The operating system was Mandrake Linux version 8.2 with Linux kernel 2.4.18. Each software implementation was compiled using gcc version 2.96 with optimization flags turned on.

Chapter 3

Related Work

This chapter will provide an overview of the work done by others towards a hardware-accelerated stochastic simulator. Typically, in the past, work in this area has focused on simulating specific biochemical models. This is the case for all related works listed below. Some groups have also introduced approximations into the SSA in favor of increasing the overall throughput of the system. The first work examined comes from Salwinski and Eisenberg, it included an approximation to the SSA. The work of Keane, Bradley, and Ebeling is considered next and it also contains an approximation to the SSA. The work of Yoshimi, Osana, Fukushima, and Amano is also considered.

3.1 Salwinski and Eisenberg's FPGA Approximation

In 2004, Lukasz Salwinski and David Eisenberg examined the use of an FPGA to exploit the highly parallel nature of information flow within biochemical networks [6,16]. They demonstrated that taking advantage of parallelism is an effective means of alleviating the high computational cost of performing stochastic simulations. However, their hardware implementation introduced approximations and was not true to Gillespie's original SSA. Furthermore, all their designs were formulated to simulate specific models. After simulating a system containing a single elementary bimolecular reaction and a system containing a simple equilibrium reaction, they tested the scalability of their approach. They were able to simulate a prokaryotic gene expression circuit (eleven coupled reactions and twelve species) while maintaining the performance seen in their previous designs. They proposed simulation rates at least an order of magnitude greater

than a software counterpart. Their work served as a proof-of-principle that reprogrammable FPGAs have the potential to efficiently simulate the stochastic behavior of biological systems. The work outlined in this paper remains mathematically equivalent to Gillespie's original SSA as well as providing a general-purpose approach to simulating a variety of chemical systems.

3.2 Keane, Bradley and Ebeling's FPGA Approximation

John Keane, Christopher Bradley, and Carl Ebeling developed an algorithm that approximates Gillespie's SSA in order to reveal a fine-grained parallel structure that is well suited to a hardware implementation [10]. At first, their team considered implementing Gibson and Bruck's Next Reaction Method [8]. However, they quickly realized the complexities involved with the algorithm would not complement the parallel capabilities of an FPGA. Since their goal was to use fine-grained parallelism to accelerate simulations, they devised a strategy that approximated Gillespie's Direct Method. They began by describing hardware to handle each reaction, allowing each reaction to be simulated simultaneously. They discretized the reaction processes in time, so reactions were only permitted at uniformly spaced discrete instants in time. A Bernoulli random process was used to approximate a Poisson process, and the probability of an event at any given discrete time step was associated with the propensity of the reaction. By utilizing a Bernoulli process to approximate the probability that a reaction will execute in a given time step, multiplications typically involved in propensity calculations could be reduced to basic compare and AND operations. The equation below represents a reaction's propensity, where X_i are discrete uniform random numbers.

$$P[X_0 < k_0] \bullet P[X_1 < S_1] \bullet P[X_2 < S_2] = kS_1S_2 \bullet \Delta t \qquad (1)$$

Although they limited their example design to a second order system, they indicated their approach would generalize to higher order systems. This strategy also eliminated the need to sum the propensities. Since each reaction's propensity was now based on the probability that the reaction would occur during a given time step, there was no longer a need to determine the next reaction executed by the system or a putative time for that matter. This approach allowed multiple independent reactions and only one dependent reaction to be performed in each time step. In the event of a collision, two or more dependent reactions occurring during a time step, the hardware paused and waited for the software to resolve the issue. Their approach was model specific and required describing, synthesizing, and routing each new design. However, they developed a compiler that read a model description in Systems Biology Markup Language (SBML) and generated a Verilog file containing the necessary modules of the system. Once a model had been prepared for the hardware it could be reused with various initial conditions. Several models of varying sizes were simulated using their FPGA approach and then compared to the performance of the same model simulated in software running the Next Reaction Method. For the largest model simulated, a system containing sixty-four species and thirty-two reactions, a speedup of 23.4 was achieved. They defined speedup based upon the average number of reaction events computed per second. However, their simulator did not capture the actual number of events and an estimate was used to determine the event rate. In addition, the event rate they assigned to their

hardware implementation neglected two sources of overhead, off-chip time step recalculations when collisions occurred and communications for data logging. They went on to reveal that the I/O communication overhead accounted for nearly 70% of the simulation time; despite this they still chose to ignore it in their speedup calculations. The work described herein is general purpose and does not require the user to redesign any hardware. In addition, the design is a statistically equivalent representation of Gillespie's SSA. Speedup values contained within this paper are based on the actual run time of the simulator.

3.3 Yoshimi, Osana, Fukushima and Amano's FPGA Simulation

Yoshimi, Osana, Fukushima, and Amano also determined that simulations of biological models often exhibit a lot of fine grain processes frequently communicating with each other. They realized that an FPGA could best utilize this fine grain parallelism inherent in biological systems [11]. To test their designs, they developed a reconfigurable platform called "ReCSiP." The ReCSiP contained a Xilinx Virtex II FPGA, and it interfaced to a host CPU via the PCI bus. To show the performance of their simulator, they modeled the Lotka system outlined in Gillespie's original paper on exact stochastic simulation [3,7]. The module designed to simulate the Lotka system consisted of two simulator modules, each containing two reactor modules, and a module to handle output control. A look up table (LUT) of logarithmic values was employed within each simulator module to allow the putative time calculation to be sped up. A portion of each reactor module contained basic steps that were relevant to any simulation executing Gillespie's SSA (putative time generation, random number generation, and reaction

selection). Therefore, this portion was applicable to any simulation on their system. However, the bulk of each reactor module in their design outlined the specifics of the model being simulated (species counts and reaction equation) and would need to be replaced with each new model. The output of each reactor module was stored in a first-in first-out (FIFO) buffer, and the output control module transferred the data to SRAM. Each reactor module appeared to be self-contained and it is unclear how species populations were coordinated across the reactor modules. Their simulation of the Lotka system were described in Verilog and could not be extended to larger chemical systems without modifying and resynthesizing several modules. They claimed it took thirty-seven clock ticks to output updated species values and fifty-two clock ticks to output the putative time. Furthermore, they claimed their reactor modules had thirty-seven pipeline stages to allow thirty-seven simulation processes to be executed in parallel. Allowing thirty-seven simulation processes to be executed in parallel may be an indication of approximations being introduced into the system, but it is not entirely clear from the paper. They declared a speedup of roughly 105 over a software implementation. However, this speedup was not based upon actual simulation run-time. They chose to compare the throughput, or simulation iterations per second, of the hardware and software. To arrive at the software throughput, they performed 500,000 reactions and timed the simulation. However, the manner in which they determined the hardware throughput is not based upon simulation time. The authors were unable to include every detail of their design and it is not clear if they included putative time generation in their speedup value. In addition, they did not specify what algorithm was implemented in software. The designs presented here are general-purpose and do not require any

redesign on the user's part. Therefore, the designs herein are applicable to larger models (within specified limits). Also, speedup values are based upon actual simulation run time.

Chapter 4

Register Based Design

4.1 Partitioning of the Problem

When the implementation of a general-purpose hardware-accelerated simulator was first considered, several questions arose. Deciding upon the most efficient SSA to implement in hardware was the first step. Since the improvements associated with the Next Reaction Method and the Optimized Direct Method are difficult to implement in hardware, these algorithms were avoided in the general-purpose hardware implementation. Gillespie's original Direct Method was the obvious choice. It offered substantial performance improvement offer the First Reaction Method, but it did not significantly complicate the hardware design. After selecting an algorithm to implement, the tasks were divided depending upon whether they should be performed in hardware or software. The FPGA handled the calculation and summation of reaction propensities in addition to the generation of a uniform random variable and determining the next reaction to execute. Both the CPU and FPGA kept a record of the species populations and updated the populations after the execution of each reaction. The CPU performed this task primarily to aid in presenting the user with data as the algorithm progressed. The generation of an exponential number and calculation of the subsequent time for the next reaction are performed in software. This was a suitable choice, since it required floating point arithmetic that is not readily available in hardware (without consuming a large portion of the available resources).

One advantage of this design was that the selected reaction and the total propensity are the only two pieces of information that the FPGA must communicate to

the microprocessor for each reaction executed. This minimized communication between the FPGA and microprocessor, alleviating what is a typical bottleneck for reconfigurable computing designs.

Another interesting advantage of the design was that the software converted all of the floating-point rate constants to integers at startup. The software read in a chemical system and found the reaction rate constant with the lowest decimal value. Then all reaction rate constants were multiplied by a factor that ensured each rate constant was an integer. Reaction rate constants were defined to be sixteen bits wide, allowing rate constant values of up to 65,535. The software alerted the user if a rate constant exceeded this limit upon adjusting it to an integer value. As long as all integer valued reaction rate constants were within limit, no error was introduced into the reaction selection process. This is true since all reaction rate constants were scaled to integers according to the lowest rate constant. Each rate constant, k_i, was scaled by F, where F was the multiplication factor needed to represent the smallest rate constant as an integer. Therefore k_i became $F*k_i$. This resulted in each reaction's propensity, a_i, becoming $F*a_i$; the total propensity becoming $F*a_{TOT}$; and the product of the total propensity and a uniform random number becoming $F*a_{TOT}*URV$. The reaction selection module still functioned properly since F could be factored out when searching the reaction propensities for the next reaction to execute. Comparing $F*a_{TOT}*URV$ to the accumulation of $F*a_i$ was equivalent to comparing $a_{TOT}*URV$ to the accumulation of a_i. This startup cost became negligible as the system was modeled over several iterations. This allowed the FPGA to be implemented using only integer logic, avoiding a floating-point arithmetic core and saving chip space.

A diagram of the division of responsibilities and communication between the software and the hardware is given in figure 4.1.

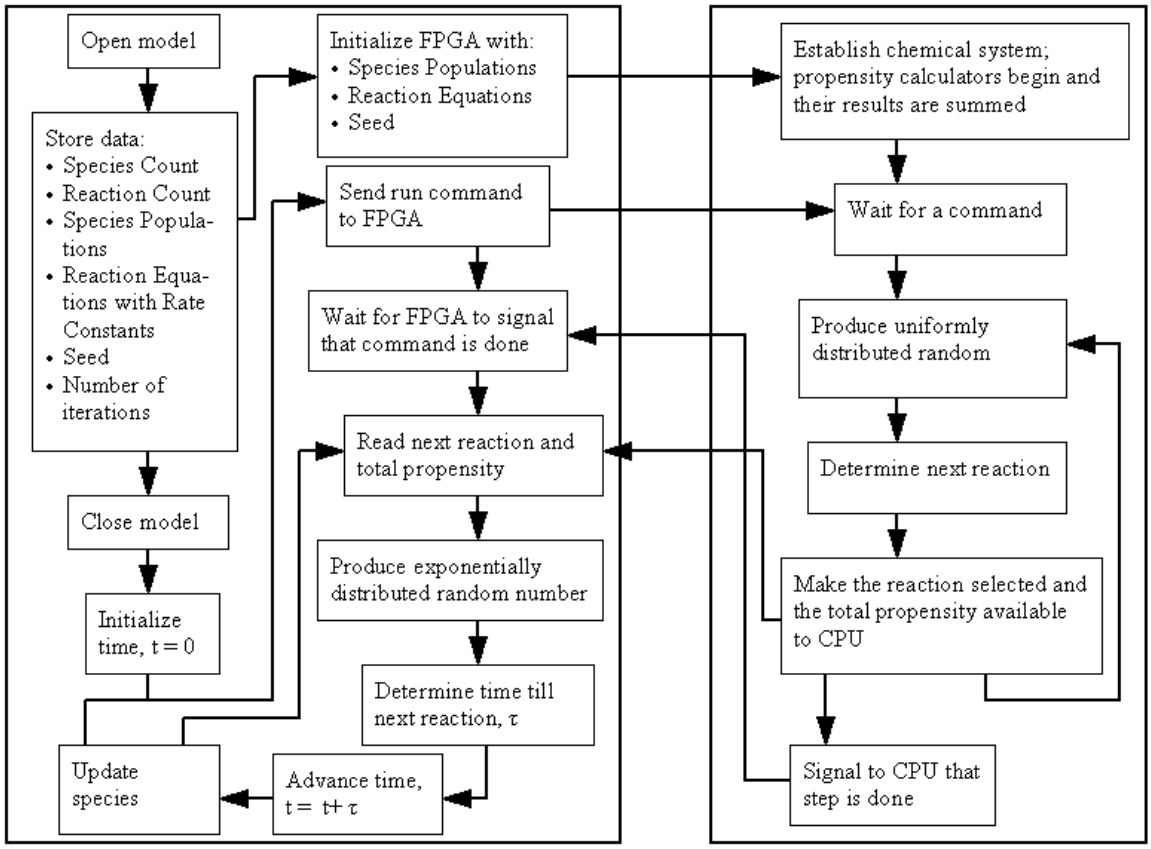

Figure 4.1 – Interaction Between Hardware and Software

4.2 Software Design

As evident in figure 4.1, the software played a complimentary role to the FPGA. It was written in C++ to allow the user to specify the model file from which to read the specifics of the biochemical system. This data was then stored in the appropriate data structures on the CPU side. Once the entire model had been loaded, the initial species populations and reaction equations were passed to the FPGA. The software facilitated the transmission of data between the FPGA and the user, in addition to allowing the user to assign various tasks to the FPGA. A command interface was developed to aid in the communication between the CPU and the FPGA during simulation. It permitted commands, as well as data, to be sent to and from the FPGA. A full description of this interface will be presented in the Hardware Design section of this Chapter.

Once a model had been fully defined in the hardware, the user could instruct the hardware to begin simulating the system. In order to alleviate the need to pass a large amount of data between the FPGA and CPU, only the selected reaction and total propensity of each iteration must be transmitted. By having the FPGA send the reaction selected, the CPU did not have to read and update the populations of all the system's species. It did require that the CPU store the species populations, and then adjust the populations of the species affected by the execution of a given reaction. The total propensity is used to generate the putative time for the iteration. The results of up to 250 iterations could be passed at a time to the CPU; this will be discussed further in the Hardware Design section of this Chapter. The CPU continued collecting results from the FPGA until the desired number of iterations has elapsed.

The software also played a crucial role in managing the time of the system. In addition to generating a putative time for each iteration of the system, it also kept track of the overall system time. The accumulation of the system time, along with the time evolution of the populations of relevant species, could be presented to the user to show a possible trajectory of the system.

4.3 Hardware Design

The first hardware implementation consisted of sixteen registers for species populations and twenty-two registers for reaction equations. All of the specifics of a given model were stored on the FPGA via flip-flops. This was not the most effective use of chip space, but it was a reasonable starting point for such a broad ranging approach.

Four of the species registers were a single bit allowing values of 0 or 1. This was an effective way of handling any on/off type reactions commonly present in chemically reacting systems. The remaining twelve species registers were twelve bits wide offering a maximum species population of 4,095. This was sufficient for most systems that meet the limited reaction specifications discussed next.

There were twenty-two modules dedicated to calculating reaction propensities in parallel; one for each of the twenty-two registers established to hold reaction equations. To minimize the chip space required for propensity calculation, there were five variations of propensity calculators. Two of the propensity calculators allowed only a single reactant of single bit-width, while eleven of the propensity calculators allowed only one reactant of any bit-width (up to twelve). Two other propensity calculators allowed for reactions with up to two reactants where one reactant is of single bit-width. In addition,

there were six propensity calculators that allowed reactions to have up to two reactants of any bit-width (up to twelve). Finally, there existed one propensity calculator that only handled the case when a species reacted with itself. All reaction equations were able to produce at most two products. It is important to note that when two of the same species reacted with one another, they were treated as separate reactants. The same is true when two of the same species were produced by a reaction. For example: $2A \rightarrow B$ was treated as $A+A \rightarrow B$, and $A \rightarrow 2B$ was treated as $A \rightarrow B+B$. Figure 4.2 shows the components of the Register Based Design and it helps to illustrate the parallelism achieved.

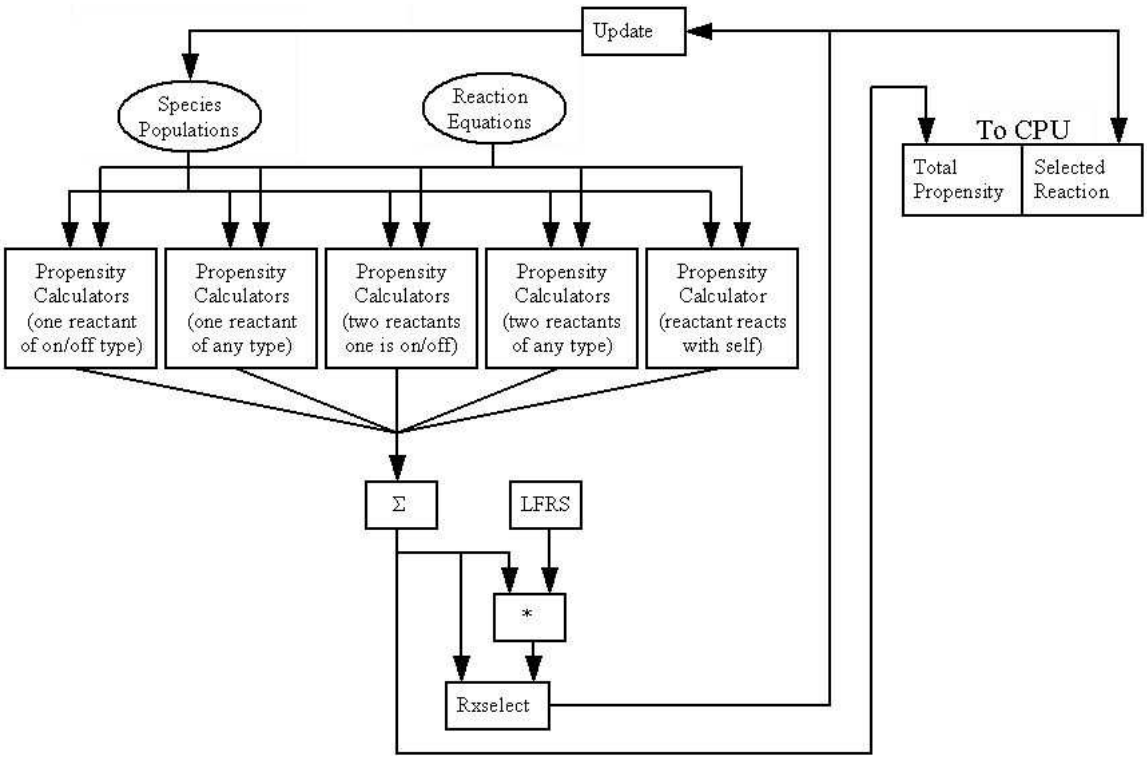

Figure 4.2 – Hardware Design Depicting Parallelism

All of the variations discussed above, concerning the propensity calculator modules, were done in an attempt to allow as many registers as possible for holding reaction equations. Different propensity calculator modules required a varying amount of input parameters and therefore necessitated dissimilar levels of complexity. This related directly to chip space; more complex propensity calculators (any species reacting with any species) consumed more gates than a simple propensity calculator (a reaction with one reactant species that is of on/off type). Since each propensity calculator was tied directly to a particular reaction equation, supporting various amounts of the different propensity calculators affected the number of total reaction equations that could be simulated. However, care had to be taken to ensure that valid biochemical systems could still be simulated. For instance, building a simulator to execute only single reactant equations would allow more total reaction equations in the system but it would not be an effective general-purpose simulator. The diverse combination of propensity calculators was chosen for this simulator in order to maximize chip space and provide sufficient resources to simulate a range of models.

Upon configuring the FPGA, a routine was executed on the host processor of the Pilchard. The routine was written in C++ and enabled the user to interact with the FPGA. The user defined a chemically reacting system and wrote it to an input file. The CPU read from this input file and sent the formatted data to the FPGA.

Reaction rate constants typically vary among the reaction equations of a chemical system. In addition, rarely are all of the rate constants integer values. In order to maximize hardware performance, floating-point arithmetic was avoided. Therefore, all reaction rate constants were converted to integer values prior to sending any data to the

FPGA. Once the model was initialized and integer values computed for the reaction rate constants, the model was passed to the FPGA. Within the FPGA, resources were laid out to compute reaction propensities, sum all propensities, generate a uniform random number, select the next reaction to execute, and update species populations. The reaction selection module sequentially searched the propensities to determine the next reaction to execute. While the update module used the reaction selection index to decrement the reaction's reactants and increment the reaction's products. The update module returned an updated value for each of the species populations, even if a species was unaffected by the execution of a given reaction. Uniform random numbers were generated by use of a linear feedback shift register (LFSR) [17].

The development of a command interface language enabled instructions and variables to be efficiently exchanged between the CPU and the FPGA and allowed for easy debugging of the hardware. This command interface language required two addresses in the DIMM interconnection of the Pilchard, one for the CPU to send commands and variables and another for the FPGA to send back data. A list of commands is given below.

1: *setspeciespop* - Sent the index and population of a species to the FPGA.

2: *readspeciespop* (debugging) - Sent the index of a species population to be read from the FPGA.

3: *setreaction* - Sent a reaction equation along with its index to the FPGA.

4: *readreaction* (debugging) - Sent the index of a reaction to be read from the FPGA.

5: *readpropensity* (debugging) - Sent the index of a propensity to be read from the FPGA.

6: *readsum* (debugging) - Read the total propensity from the FPGA.

7: *setseed* - Sent the seed for the uniform random number generation on the FPGA.

8: *readURV* (debugging) - Read the uniform random number generated on the FPGA.

9: *nextURV* (debugging) - Instructed the linear feedback shift register (LFSR) on the FPGA to generate a new uniform random variable.

10: *readproduct* (debugging) - Read the product of the uniform random number times the total propensity.

11: *readrxselected* (debugging) - Read the index of the next reaction to be executed.

12: *updatespecies* (debugging) - Updated the species populations on the FPGA according to the next reaction to be executed.

13: *step* - Instructed the FPGA to determine and execute 250 reactions. This command is discussed in detail later.

Some commands listed above were developed for preliminary debugging purposes; this is indicated in the command descriptions above. These debugging commands were removed from the final version in order to maximize the number of gates available to define a chemical system. The commands listed above are crucial for interacting with the FPGA to model a chemical system. The FPGA interacted with the host processor via a DIMM interface. In order to read or write to the DIMM, the FPGA defined an eight bit wide address in the DIMM. This allowed for 256 separate

addressable locations in the DIMM. Each of these addresses could hold sixty-four bits of data. Refer to Chapter 2 for further description of the hardware platform.

The step command was used to complete the iterations of a system model, so a more in depth view is provided below. When the CPU issued a step command, pertinent data for 250 iterations was placed into 250 addresses of the DIMM following the selection of each executed reaction by the FPGA. Each address contained the reaction selected along with the total propensity prior to the FPGA executing the reaction. This command is repeated until the desired number of iterations is reached. The step command executed 250 reactions in order to fully utilize the portion of the DIMM addressable by the FPGA. Upon completing an instruction from the CPU, the FPGA cleared the command from the DIMM address. The CPU waited for this to occur indicating that the FPGA is finished. If the command was a step, the CPU cycled through DIMM addresses from 0x2 to 0xFB. At each address, the CPU used the total propensity to calculate an exponential random variable and a time until the next reaction. The species values stored on the CPU were then updated according to the reaction index at the address. The CPU then continued issuing the user's commands to the FPGA. Step commands were repeated until all iterations required of the system have been executed.

4.4 Comparison of Results

In order to test this hardware-accelerated approach to exact stochastic simulation, several exact stochastic simulation algorithms were utilized in software. The algorithms chosen for comparison were: Gillespie's First Reaction Method, Gillespie's Direct Method, Gibson and Bruck's Next Reaction Method, and Cao, Li, and Petzold's

Optimized Direct Method. All of the software implementations were from pre-existing designs developed by James McCollum [18]. Each software algorithm was compiled and executed on the Pilchard's host processor, discussed in Chapter 2. Each algorithm was compiled using gcc version 2.96 with optimization flags turned on. The performance of each, when given identical chemically reacting systems, was compared to the hardware version. In the following tables, the hardware version is labeled "Hardware Direct." Speedup values were calculated by dividing the execution time of the software method by the execution time of the accelerated hardware method. Two actual biochemical systems were chosen to use as models to calculate the resulting speedup of the hardware implementation.

The first chemical system considered was an auto-regulated gene expression model based on the work of Simpson et al [2]. The system contained ten species and fourteen reaction equations. The initial species populations and reactions are given in figure 4.3.

Chemical Species:

```
     gene1 = 1
  gene1off = 0
     gene2 = 0
   gene2on = 1
     mRNA1 = 3
     mRNA2 = 3
        P1 = 44
        P2 = 44
    Dimer1 = 19
    Dimer2 = 19
```

Chemical Reactions:

```
           gene1 → gene1 + mRNA1        k = 120
           mRNA1 → *                    k = 30
           mRNA1 → mRNA1 + P1           k = 13
              P1 → *                    k = 1
         P1 + P1 → dimer1               k = 518
          dimer1 → P1 + P1              k = 51813
  dimer1 + gene2 → gene2on              k = 518
         gene2on → gene2 + dimer1       k = 51813
         gene2on → gene2on + mRNA2      k = 130
           mRNA2 → *                    k = 30
           mRNA2 → mRNA2 + P2           k = 13
              P2 → *                    k = 1
         P2 + P2 → dimer2               k = 518
          dimer2 → P2 + P2              k = 51813
```

Figure 4.3 – Self Regulated Model [2]

Execution times for each method simulating this self-regulating system for 100,000,000 iterations is given in Table 4.1 along with the associated speedup achieved by the hardware implementation.

Table 4.1 – Speedup Associated with Self Regulated Model

Method	**Execution Time**	**Speedup**
Hardware Direct	77.798	—
First Reaction	814.033	10.46
Direct	225.114	2.89
Next Reaction	174.656	2.24
Optimized Direct	109.410	1.41

The second system considered was a model of genomically based oscillation, based on two mutually interacting genes. This model comes from Vilar et al [12]. An activator provided positive feedback to the system, while a repressor provided negative feedback. The system contained nine species and sixteen reaction equations. The initial species populations and reactions are given in figure 4.4.

Chemical Species:

```
    A = 0
    C = 0
   DA = 1
  DAp = 0
   DR = 1
  DRp = 0
   MA = 0
   MR = 0
    R = 0
```

Chemical Reactions:

```
      DA → DA + MA          k = 50
     DAp → Dap + MA         k = 500
      DR → DR + MR          k = 0.01
     DRp → DRp + MR         k = 50
      MA → A + MA           k = 50
      MR → MR + R           k = 5
  A + DA → Dap              k = 1
   A + R → C                k = 2
  A + DR → DRp              k = 1
       A → *                k = 1
       C → R                k = 1
      MA → *                k = 10
      MR → *                k = 0.5
       R → *                k = 0.2
     DAp → A + DA           k = 50
     DRp → A + DR           k = 100
```

Figure 4.4 – Genomically Based Oscillation Model [12]

Execution times for each method simulating this genomically based oscillation model for 100,000,000 iterations is given in Table 4.2 along with the associated speedup achieved by the hardware implementation.

Table 4.2 – Speedup Associated with Genomically Based Oscillation Model

Method	Execution Time	Speedup
Hardware Direct	78.259	—
First Reaction	805.044	10.29
Direct	230.558	2.95
Next Reaction	252.125	3.22
Optimized Direct	118.948	1.52

It is clear that simulating chemically reacting systems on a reconfigurable computing platform provides a speedup over any of the methods executed in software. Furthermore, these results illustrate that employing FPGAs in stochastic simulation is an effective way to accelerate the simulation of useful biological systems.

4.5 Difficulties and Design Limitations

The register based hardware approach proved to be an effective way of accelerating exact stochastic simulation. However, the design does contain some inefficiencies and limitations. The primary bottlenecks of the design are processing within the CPU and communication between the FPGA and CPU. Having the CPU

generate an exponentially distributed random number and the putative time is detrimental to performance. Since this cost is also associated with software implementations, it represents an area where speedup could be achieved in hardware. This design also severely restricts the size of systems that can be simulated. By limiting the system to only sixteen species and twenty-two reactions, it is difficult to find biologically relevant models to simulate. The ability to handle more reactions in hardware could greatly improve performance. This is partly evident by the results shown within this chapter. It is feasible that the speedup value for the hardware implementation would increase further by modeling a system that fully utilizes the resources laid out in the hardware design. This is due to the hardware's ability to perform several operations in parallel. For biochemical systems that meet the system parameters (i.e. twenty-two reaction equations), the hardware exhibits a steady performance. However, software is not able to scale in such a manner and performs steps sequentially. In general, the performance of a software implementation of the SSA declines as more reaction equations are introduced.

A summary of the device utilization can be seen in figure 4.5. In addition, a view of the timing constraints can be found in figure 4.6. It is clear that the current constraints of the register-based approach completely consume the chip space of the FPGA. Therefore, there is no room to address the issues described above. More species and reactions cannot be stored in hardware, and exponential random number generation must remain a task for the CPU. This is the motivation for a second approach to a general-purpose hardware accelerated SSA. Blocks of Random Access Memory (BRAM) are available on the Pilchard reconfigurable platform. These will allow the hardware to accept larger sized systems while, at the same time, reducing the chip space so

exponential random numbers can be generated in hardware. These techniques will be further discussed in Chapter 5.

Number of External GCLKIOBs	1	out of 4	25%
Number of External IOBs	104	out of 158	65%
Number of LOCed External IOBs	104	out of 104	100%
Number of BLOCKRAMs	4	out of 96	4%
Number of SLICEs	12091	out of 12288	98%
Number of DLLs	1	out of 8	12%
Number of GCLKs	3	out of 4	75%
Number of STARTUPs	1	out of 1	100%

Figure 4.5 – Register Based Approach Design Utilization Summary

Constraint	Requested	Actual	Logic Levels
NET "dimm_ck_bufg" PERIOD = 7.519 nS HIGH 50.000000 %	N/A	N/A	N/A
* PERIOD analysis for net "clkdllhf_clk0" derived from NET "dimm_ck_bufg" PERIOD = 7.519 nS HIGH 50.000000 %	7.519ns	16.357ns	1
* PERIOD analysis for net "clkdllhf_clkdiv" derived from NET "dimm_ck_bufg" PERIOD = 7.519 nS HIGH 50.000000 %	37.595ns	127.663ns	53

Figure 4.6 – Timing Constraints of Register Based Design

Chapter 5

Block RAM Based Design

5.1 Partitioning of the Problem

After completing the Register Based Design, a second pass at a general-purpose hardware accelerated exact stochastic simulator was attempted. Two limitations of the Register Based Design were focused on: (1) moving more of the algorithm into hardware, primarily the generation of exponentially distributed random numbers and (2) simulating larger models that contain more reaction equations. Having already developed a working knowledge of the Pilchard Reconfigurable Platform [5], it was used in the second design as well. Since the previous design consumed all of the available chip space on the FPGA, a new approach required a better utilization of the resources available. The blocks of random access memory (BRAM) present on the Pilchard board were a promising solution to optimize the use of the chip space available. Storing species populations, reaction equations, and reaction propensities in BRAM would allow larger biochemical systems to be modeled while allowing more of the SSA to be defined in hardware. Gillespie's original Direct Method was still adequate to the design goals and was used in the second approach. Much of the overall design remained the same as the Register Based Design discussed in Chapter 4. All the tasks and interfacing were the same, except the CPU calculated the putative time for each reaction based upon an exponential random number generated by the FPGA.

This design still required that the selected reaction and the total propensity be transmitted to the CPU from the FPGA. However, the FPGA now had to transmit an exponential random number for each iteration of the system as well. Just as with the

Register Based Design, the software converted all of the floating-point rate constants to integers at startup to avoid any floating arithmetic on the FPGA. A diagram of the division of responsibilities and communication between the software and the hardware is given in figure 5.1. The layout was very similar to the previous design with the exception that exponential random number generation was done on the FPGA.

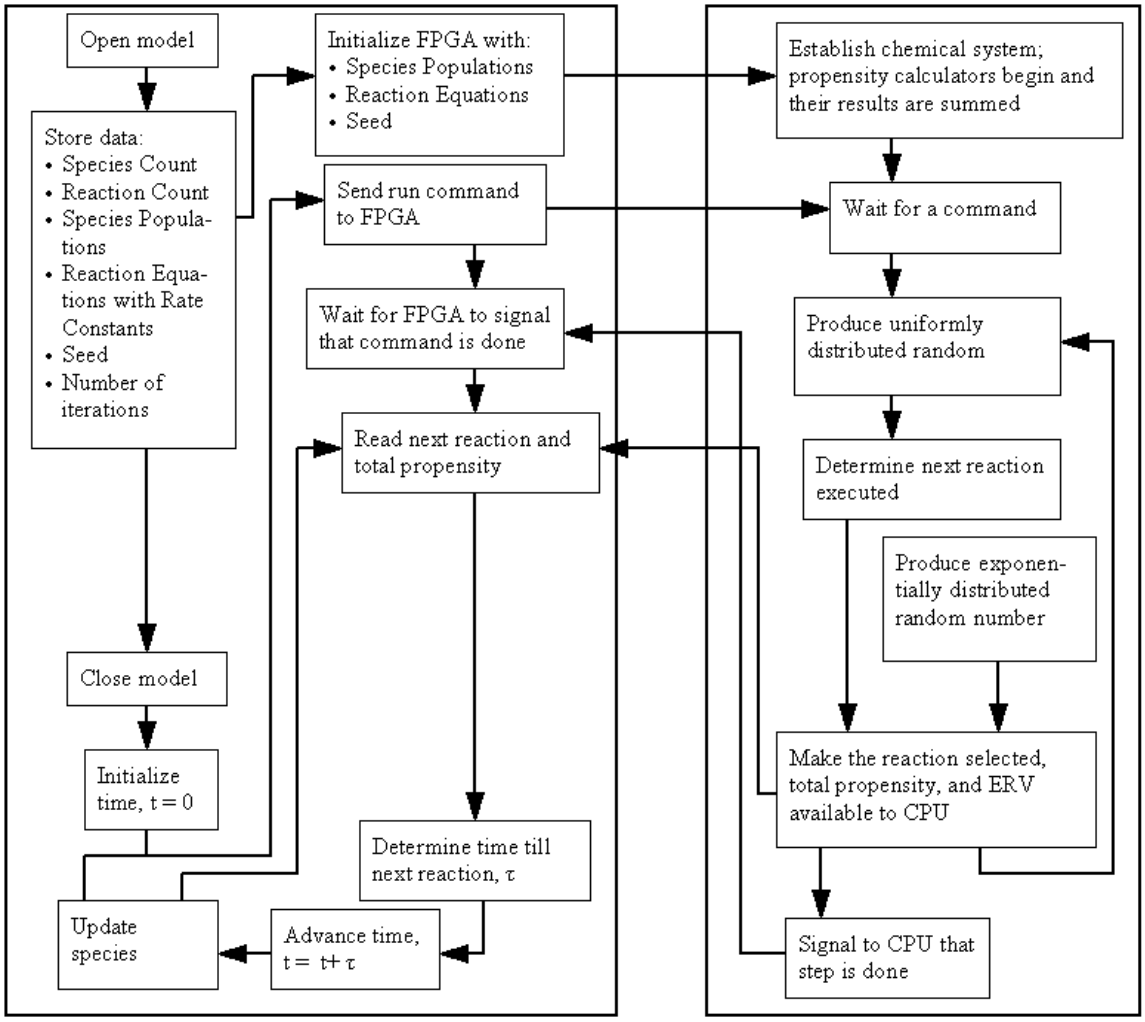

Figure 5.1 – Interaction Between Hardware and Software

5.2 Software Design

The software for this implementation was very similar to that of the previous design. Some modifications were made, written in C++, to allow larger models and to read exponential random numbers from the FPGA. Refer to the Software Design section of Chapter 4 for a thorough overview of the software's role. Since the BRAM design required more information be passed from the FPGA to the CPU with each iteration, the information (total propensity, next reaction, and an exponential random number) from up to 125 iterations could be passed at a time to the CPU. The CPU's behavior in this design mimicked that of the previous design, and results were collected from the FPGA until the desired number of iterations had elapsed.

By moving the exponential random number generation to the FPGA, another modification had to be made to the software. When generating a putative time for each iteration of the system, the CPU read the exponential random number from the FPGA and converted it from fixed point to floating point before dividing by the total propensity to find the time till the next reaction.

5.3 Hardware Design

The BRAM available on the Virtex 1000E FPGA was formatted to allow systems with a maximum of 127 species populations and sixty-three reaction equations to be simulated. In addition, the BRAM was formatted to hold the resulting propensity associated with each reaction equation. In stark contrast to the Register Based Design, all species populations were sixteen bits wide. This led to a maximum species population of 65,535. The restrictions placed on defining a reaction equation were also lifted. That is

to say each reaction equation was able to consist of up to two reactants and two products, and the bit width of the reaction rate constant remained at sixteen. This removed the need to have several variations of propensity calculators. For the BRAM Based Design, each propensity calculator was built to be general-purpose and able to handle any of the cases discussed in the Register Based Design.

Allowing more reaction equations to be simulated did introduce a substantial obstacle to maximizing performance. Whereas in the Register Based Design each reaction equation had its own dedicated propensity calculator, the number of possible reaction equations in the BRAM Based Design prohibited such an approach. The chip space required to instantiate sixty-three propensity calculators exceeded that which was available on the Pilchard. The use of general-purpose propensity calculators and the introduction of other complications allowed eight propensity calculators to fit into the available space. Therefore, each propensity calculator had to calculate the propensity for up to eight reaction equations. This was an obvious drain on performance. However, it did have a favorable effect as well. As each propensity calculator cycled through its eight reaction equations, it accumulated the calculated propensities. It stored the sum of the propensities in registers after the first four reaction equations and after all eight reaction equations. This led to a total of sixteen registers, two per propensity calculator, being used to store these *Partial Sums*. Not only did this simplify the process of determining the total propensity (of all sixty-three reactions), it also aided in the selection of the next reaction.

The use of the partial sums allowed the reaction selection module to operate as a mock search tree. Figure 5.2 will give an idea of how they would be useful.

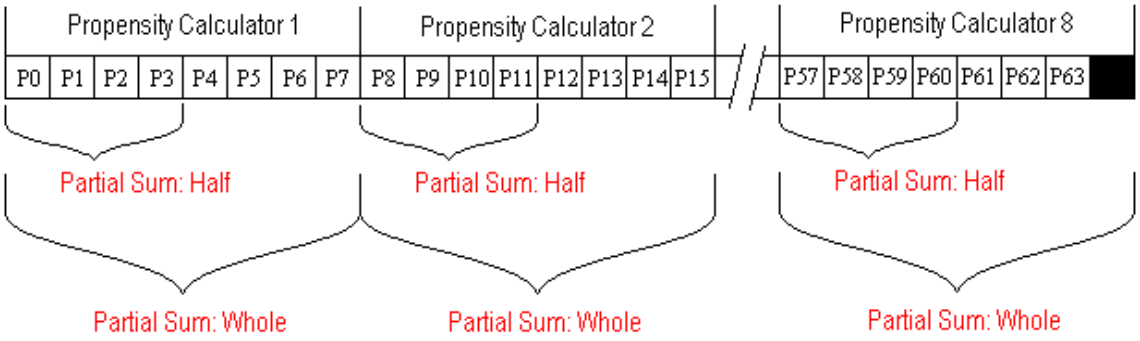

Figure 5.2 – Partial Sums Use in Selecting Next Reaction

Under the timing constraints seen in figure 5.5, it took two clock ticks to read data from BRAM, therefore it would be devastating if every propensity had to be read sequentially in order to determine the next reaction. The partial sums helped the reaction selection module narrow down the search to four propensities in BRAM. This helped offset the cost of having each propensity calculator find the propensity of eight reaction equations.

To update the species populations, the index of the selected reaction was used to read the corresponding reaction equation from BRAM to determine which species populations were affected. The indices of the reactants and products were then used to find their populations, prior to execution of the reaction, in BRAM. Registers were used to hold the reactant species populations after being decremented and the product species populations after being incremented. These registers were then used to update the species populations of the appropriate BRAM. The populations of reactants and products were updated sequentially. At each stage of the update, the affected species population registers were updated to reflect the any change in pertinent species populations. Therefore if a species reacted with itself, its corresponding population in BRAM would be decremented twice. This would also handle the case when a reaction produced two molecules of a given species, thus incrementing a population twice. In addition, it makes it possible to effectively handle reaction equations that contain a species that serves as a reactant as well as a product. In that situation, the species population will remain the same.

The Intellectual Property (IP) block used to generate exponentially distributed random numbers was developed by James McCollum [17]. It used an LFSR and a look up table to interpolate a value along an exponential curve.

Figure 5.3 shows the components of the BRAM Based Design and it helps to illustrate the parallelism achieved.

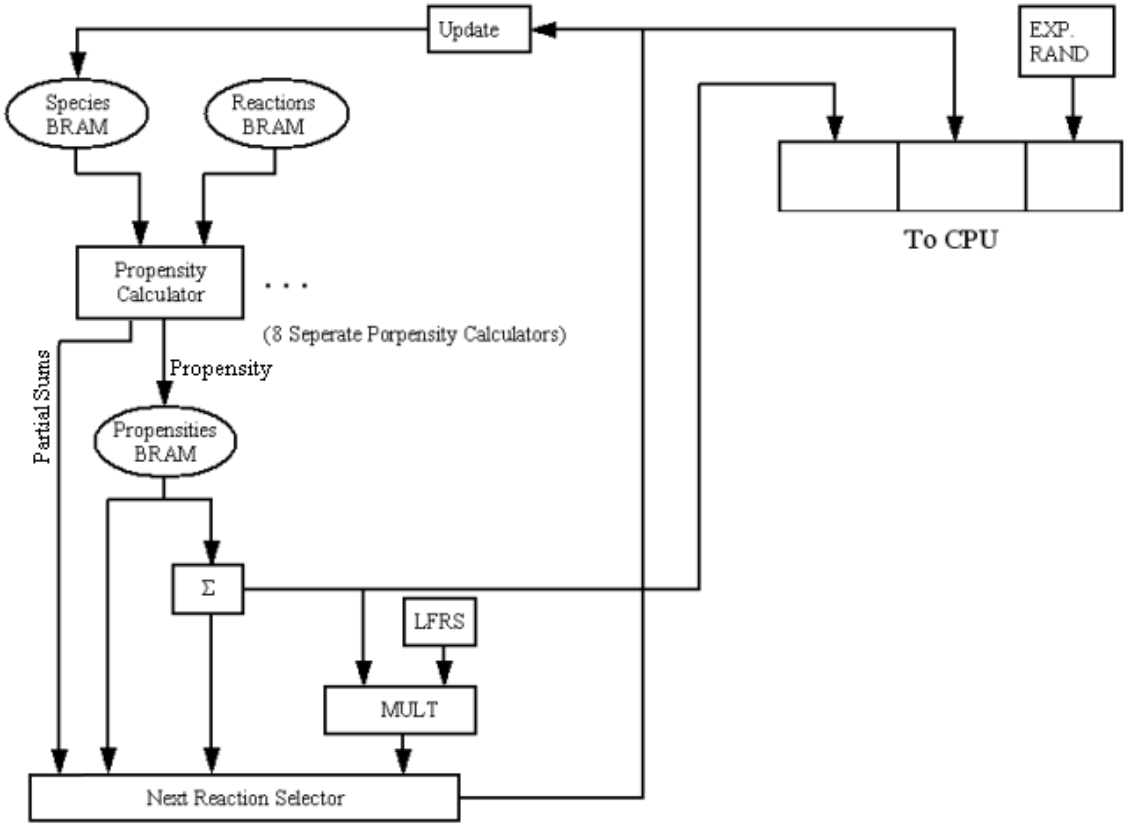

Figure 5.3 – BRAM Based Design Depicting Parallelism

The user interacted with the FPGA in the same manner as previously described in the Register Based Design. A routine, written in C++, enabled the user to define a chemically reacting system for the CPU to read and then send the formatted data to the FPGA. Again, all reaction rate constants were converted to integer values prior to sending any data to the FPGA in order to avoid floating point arithmetic. Within the FPGA, resources were laid out to compute reaction propensities, sum all propensities, generate a uniform random number, select the next reaction to execute, generate an exponential random number, and update species populations.

The command interface language developed for the Register Based Approach was reused with some modification. The command interface language required two addresses in the DIMM interconnection of the Pilchard, one for the CPU to send commands and variables and another for the FPGA to send back data. A list of valid commands for the BRAM Based Design is given below.

1: *setSP* - Sent the index and population of a species to the FPGA.
2: *readSP* (debugging) - Sent the index of a species population to be read from the FPGA.
3: *setRX* - Sent a reaction equation along with its index to the FPGA.
4: *readRX* (debugging) - Sent the index of a reaction to be read from the FPGA.
5: *readPROP* (debugging) - Sent the index of a propensity to be read from the FPGA.
6: *readPSUM* (debugging) - Read any of the partial sums generated by the propensity calculators, can also read total propensity from the FPGA.

7: *setseed* - Sent the seed for the uniform random number generation on the FPGA.

8: *readURV* (debugging) - Read the uniform random number generated on the FPGA.

9: *newURV* (debugging) - Instructed the linear feedback shift register (LFSR) on the FPGA to generate a new uniform random variable.

10: *readPRODUCT* (debugging) - Read the product of the uniform random number times the total propensity.

11: *readSELECTION* (debugging) - Read the index of the next reaction to be executed.

12: *readERV* (debugging) - Read the exponential random number generated on the FPGA.

13: *initPROP* (debugging) - Used to initiate the propensity calculators after loading details of the system.

14: *step* - Instructed the FPGA to determine and execute 250 reactions. This command is discussed in detail later.

Many commands listed above were developed for debugging purposes; this is indicated in the command descriptions above. Many debugging commands were removed from the final version in order to maximize the number of gates available to define a chemical system.

Recall from Chapter 4 that the FPGA interacts with the host processor via a DIMM interface with 256 separate addressable locations, each with sixty-four bits of data. The step command of the BRAM version is similar to the Register Based Design.

However, in this case, data for only 125 iterations is placed into DIMM. This is because each iteration now required two addresses since the exponential random number is thirty-two bits wide, the total propensity is clipped to thirty-two bits, and six bits were needed for the selected reaction remain to be sent. Each iteration was given two addresses of the DIMM, therefore the 250 addresses of the DIMM that are readable by the CPU allowed the step command to perform 125 iterations at a time.

Just as in the previous design, the FPGA cleared the command from the DIMM address upon completing an instruction from the CPU. The CPU waited for this to occur indicating that the FPGA is finished. If the command was a step, the CPU cycled through DIMM addresses from 0x2 to 0xFB in pairs. The first address contained the total propensity and the exponential random number. The CPU used this information to compute a time until the next reaction. The second address contained the next reaction to be executed. The species values stored on the CPU were then adjusted according to the reaction index given. The CPU then continued issuing the user's commands to the FPGA. Step commands were repeated until all iterations required of the system had been executed.

5.4 Comparison of Results

The BRAM Based Design will be compared to the following algorithms implemented in software: Gillespie's First Reaction Method, Gillespie's Direct Method, Gibson and Bruck's Next Reaction Method, and Cao, Li, Petzold's Optimized Direct Method, and McCollum's Sorting Direct Method. Once again, these were pre-existing software designs developed by James McCollum [18,20]. Two models that meet the

criteria of the BRAM Based Design were chosen. Using the above software implementations to simulate these systems generated the following results. The results of the hardware-accelerated simulator are labeled "Hardware Direct." Each software algorithm was compiled and executed on the Pilchard's host processor, discussed in Chapter 2. Each algorithm was compiled using gcc version 2.96 with optimization flags turned on. Two real biological models were used to test the BRAM Based Design. Dr. Chris Cox at the University of Tennessee formulated each model [18]. The SBML description of each model can be found in Appendix E as well as an outline of the contents of an SBML model.

The first chemical system considered was a simple model of a gene whose protein undergoes dimerization. This model consisted of eight species and thirteen reaction equations. The execution time of each method simulating the above system for 1,000,000 iterations is given in Table 5.1.

Table 5.1 – Speedup Associated with Protein Dimerization

Method	**Execution Time**	**Speedup**
Hardware Direct	3.300	—
First Reaction	19.250	5.83
Direct	6.670	2.02
Next Reaction	4.896	1.48
Optimized Direct	4.036	1.22
Sorting Direct	3.386	1.03

The third chemical system modeled was tuberculosis. It consisted of seventeen species and twenty-three reaction equations. The execution time of each method simulating the above system for 1,000,000 iterations is given in Table 5.2.

Table 5.2 – Speedup Associated with for Tuberculosis

Method	**Execution Time**	**Speedup**
Hardware Direct	3.32	—
First Reaction	36.669	11.04
Direct	10.254	3.09
Next Reaction	10.656	3.21
Optimized Direct	4.355	1.31
Sorting Direct	4.291	1.29

In order to fit into the constraints of the hardware implementation, some modifications were made to the original tuberculosis SBML file. The original system contained a reaction equation that produced more than two products. Therefore in order for the model to fit into the hardware implementation, a dummy species and a dummy reaction equation had to be established. When a reaction occurred with numerous products, a dummy species would be activated. This dummy species would be associated with a dummy reaction equation with an extremely high rate constant, in order to be reasonably certain that the reaction would occur next. The dummy reaction equation would then increment product species populations or possibly activate another dummy

species/reaction equation. This is a departure from the biologically relevance of the model, but it is effective at demonstrating the performance of the hardware implementation. The original SBML file for Tuberculosis, along with the modified version, can be found in Appendix E.

Once again, the results from the two models simulated above suggest that the speedup achieved by the hardware implementation generally increases as the number of reaction equations increases. This is due to the fact that resources have been laid out in hardware to handle any system within the parameters specified earlier. The hardware's execution time is relatively steady while the workload of a software-implemented method will usually increase with additional reaction equations.

5.5 Difficulties and Design Limitations

Although the BRAM Based Design served to address some of the shortcomings of the Register Based Design, it also introduced some new limitations.

The speedup achieved by the BRAM Based Design is below the speedup achieved by the Register Based Design. However, in terms of future development of a hardware design, the BRAM Based Design offers the most potential. Storing species populations and reaction equations in BRAM allowed large systems to be simulated, but accessing BRAM was time consuming. Reading an address in BRAM required two clock ticks: (1) to set the address from which to read (2) to read the data from BRAM. This slowed down performance due to the number of times BRAM must be read when calculating a propensity. Simulating larger systems was naturally more complex. As discussed earlier, modeling systems that contain up to sixty three reactions will not allow each reaction to

have its own dedicated reaction propensity calculator running in parallel. The solution to this problem was to have eight propensity calculators running in parallel, each one calculating the propensity of eight reaction equations. This limited performance since each propensity calculator had to sequentially compute the propensity of eight reaction equations. The ability to model systems with more reaction equations also required more time for the reaction selection stage. In the Register Based Design, propensities were searched sequentially to find the next reaction to execute. This line of attack was not as appealing when dealing with a system with sixty-three reactions. The partial sums generated by the propensity calculators discussed in the Hardware Design section of this chapter were an attempt to improve the search time required by the reaction selection module, but it remained a time consuming process.

Moving the exponential random number generation to the FPGA reduced the amount of processing done by the CPU when calculating the putative time. However, it required more information to be passed from the FPGA to the CPU. As mentioned previously throughout this paper, communication between the FPGA and the CPU is a typical bottleneck of this design. The CPU now not only had to compute the putative time via floating point arithmetic, but it also had to convert the exponential random number from fixed point to floating point notation. This was not as detrimental to performance as the introduction of more reaction equations, but it did prohibit maximizing the potential gain of moving the exponential random number generation into hardware. In order to fully utilize the potential speedup of performing exponential random number generation in hardware, the inclusion of a floating-point core could be added to the FPGA. This would allow the FPGA to not only calculate the putative time,

but also maintain the system time. This is discussed more in section 6.2 as a suggestion for future work. Figure 5.4 depicts the device utilization summary of the BRAM Based Design, while figure 5.5 illustrates the time constraints on the design.

```
Device utilization summary:

   Number of External GCLKIOBs         1 out of 4      25%
   Number of External IOBs           104 out of 158    65%
      Number of LOCed External IOBs  104 out of 104   100%

   Number of BLOCKRAMs                67 out of 96     69%
   Number of SLICEs                 9507 out of 12288  77%

   Number of DLLs                     1 out of 8      12%
   Number of GCLKs                    3 out of 4      75%
   Number of STARTUPs                 1 out of 1     100%
```

Figure 5.4 – BRAM Based Design Utilization Summary

Constraint	Requested	Actual	Logic Levels
NET "dimm_ck_bufg" PERIOD = 7.519 nS HIGH 50.000000 %	N/A	N/A	N/A
* PERIOD analysis for net "clkdllhf_clk0" derived from NET "dimm_ck_bufg" PERIOD = 7.519 nS HIGH 50.000000 %	7.519ns	13.684ns	1
* PERIOD analysis for net "clkdllhf_clkdiv" derived from NET "dimm_ck_bufg" PERIOD = 7.519 nS HIGH 50.000000 %	37.595ns	91.619ns	55

Figure 5.5 – Timing Constraints of BRAM Based Design

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This work has explored the possibility of a general-purpose hardware accelerator for stochastic simulation. Two hardware-based approaches were given. Both approaches, the Register Based Design and the BRAM Based Design, were shown to offer a speedup over several stochastic simulation algorithms implemented in software. Although the hardware designs outperformed all the software versions by up to 1.5X, work still remains to develop an optimized hardware version. The topics listed below for future work could direct new hardware designs towards an optimized solution. This could allow biological researchers to accurately model biochemical systems in order to develop the gene therapy and drugs of tomorrow.

The use of FPGAs to accelerate the simulation of biological models appears to be a plausible option. This work, along with related work in Chapter 3, has shown that FPGAs can play an important part towards speeding up the simulation times of biological models. However, a general-purpose hardware design is essential in order for Biologists to consider using an FPGA for stochastic simulations. This was precisely the goal of the research presented. This work can now serve as a foundation upon which future general-purpose designs will undoubtedly achieve superior performance and empower Biologists to accurately and quickly simulate biological models.

6.2 Hardware Improvements

Performance improvement could be obtained by porting either of the hardware designs outlined within this paper, the Register Based Design or the BRAM Based Design, to an updated reconfigurable computing platform. One such platform now available at the University of Tennessee, Knoxville is the Amirix AP130 [19]. This development board contains a Xilinx Virtex II Pro XC2VP30 FPGA. The Virtex II Pro contains roughly the same number of gates as the Virtex 1000E, but it does have advantages. Within the Virtex II Pro exist two IBM Power PC (PPC) 405 cores tightly coupled with the FPGA. However, the PPC do not support floating point operations so putative time generation will still be a hindrance to performance. The AP130 contains sixty-four MB of Synchronous Dynamic Random Access Memory (SDRAM) onboard in addition to 136 dedicated eighteen-bit multipliers on the Virtex II Pro [15]. Although the AP130 communicates with the host via the PC's PCI bus, the above enhancements make the AP130 a worthy candidate for future endeavors.

Another hardware improvement might be to employ a larger FPGA. Doing so with the Register Based Design would provide for more reaction equations to be simulated, however routing might become an issue and require a different approach. Implementing the BRAM Based Design onto a larger FPGA might be worth the effort. Increasing the available number of gates would allow more propensity calculators to be running in parallel, reducing the time needed to calculate all the reaction propensities. In addition, it might facilitate the inclusion of a floating point IP block onto the FPGA. This would allow all putative time and even accumulated system time to be calculated on the FPGA. The ability to perform putative time calculations and maintain the system time on

the FPGA could allow the user to define a time interval for printing results as well as a time to end the simulation. Then the FPGA would only have to communicate with the CPU after each time interval. Furthermore, the species populations would not have to be tallied on the CPU side since the FPGA could transmit them along with the system time after each time interval.

6.3 Design Improvements

Each of the hardware designs explored during this paper had design choices that played a subtle role in the resulting performance. For the Register Based Design, it might be beneficial to alter the reaction selection module from a sequential search. This could be done in a fashion similar to that performed in the BRAM Based Design. However, with only twenty-two reaction equations supported, a sequential reaction selection module is not a horrible choice. Furthermore, the cost of ranking the reaction propensities might outweigh the gain associated with implementing a cleverer search routine. A design improvement for the BRAM Based Design might be to include a variety of propensity calculators as was done in the Register Based Design. This might release some of the chip space on the FPGA allowing more propensity calculators to be implemented thus reducing the time it takes to calculate all propensities.

6.4 Algorithm Improvements

Improved stochastic simulation algorithms continue to surface. Primarily the improvements are directed towards software implementations. However, future algorithms may introduce enhancements that are readily adaptable to hardware. In

addition, some components of previous algorithms could play a positive role in hardware acceleration of stochastic simulations. For instance, the use of a dependency graph [8] might have a positive impact on performance.

6.5 Application Specific Integrated Circuit Design

As one would expect, moving either of the designs mentioned within this paper to an application specific integrated circuit (ASIC) would improve performance. Porting the design to an ASIC would offer a substantial increase in clock frequency, resulting in improved speedup values, as well as a dramatic increase in the number of available gates. However, production of an ASIC is not currently a feasible choice. Aside from being very expensive to fabricate, they are not suitable to evolving designs [13]. Once a design is implemented in an ASIC, it is permanent. This complicates the choice to utilize an ASIC for a design. The designs presented herein would benefit from the increased clock frequency, but they would not maximize the chip space offered by an ASIC. In order to do so, considerable testing and debugging of a design would be required. There is promise in an ASIC design, but implementing the SSA in hardware is still a relatively new strategy and several obstacles remain before constructing a fully optimized hardware solution on an ASIC.

REFERENCES

References

[1] C. V. Rao, D. M. Wolf, and A. P. Arkin, "Control, Exploitation, and Tolerance of Intracellular Noise," *Nature*, vol. 420, pp. 231-237, Nov 2002.

[2] M. L. Simpson, C. D. Cox, and G. S. Sayler, "Frequency Domain Analysis of Noise in Auto-regulated Gene Circuits," *Proceedings of the National Academy of Sciences*, vol. 100, pp. 4551-4556, Apr 2003.

[3] D. T. Gillespie, "General Method for Numerically Simulating Stochastic Time Evolution of Coupled Chemical Reactions," *Journal of Computational Physics*, vol. 22, pp. 403-434, 1976.

[4] D. Endy and R. Brent, "Modeling Cellular Behavior," *Nature*, vol. 409, pp. 391-395, 2001.

[5] K. H. Tsoi, *Pilchard User Reference (v0.2)*, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong, 2003.

[6] L. Salwinski and D. Eisenberg, "In silico simulation of biological network dynamics," *Nature Biotechnology*, vol. 22, pp. 1017-1019, August 2004.

[7] D. T. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions," *Journal of Physical Chemistry*, vol. 81, pp. 2340-2361, 1977.

[8] M. Gibson and J. Bruck, "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," *Journal of Physical Chemistry A*, vol. 104, pp. 1876-1889, 2000.

[9] Y. Cao, H. Li, and L. R. Petzold, "Efficient Formulation of the Stochastic Simulation Algorithm for Chemically Reacting Systems," *Journal of Chemical Physics*, vol. 121, pp. 4059-4067, Sep 2004.

[10] J. F. Keane, C. Bradley, and C. Ebeling, "A Compiled Accelerator for Biological Cell Signaling Simulations," *FPGA 2004*, pp. 233-241, 2004.

[11] M. Yoshimi, Y. Osana, T. Fukushima, H. Amano, "Stochastic Simulation for Biochemical Reactions on FPGA," *Field-Programmable Logic and Applications, Proceedings Lecture Notes in Computer Science*, vol. 3203, pp. 105-114, 2004.

[12] J. Vilar, H. Kueh, N. Barkai, S. Leibler, "Mechanisms of Noise Resistance in Genetic Oscillators," *PNAS*, vol. 99, pp. 5988-5992, 2002.

[13] D. Bouldin, (2003). *Overview of FPGAs and ASICs*, http://vlsi1.engr.utk.edu/ece/bouldin_courses/private_html/1-overview-fpga-asic-color.pdf, (accessed July 2005).

[14] *Xilinx Virtex XCV1000E*; Data Sheet; Xilinx: San Jose, CA, (June 16) 2004, http://www.xilinx.com/bvdocs/publications/ds022.pdf, (accessed July 2005).

[15] *Xilinx Virtex XC2VP30*; Data Sheet; Xilinx: San Jose, CA, (June 20) 2005, http://www.xilinx.com/bvdocs/publications/ds083.pdf, (accessed July 2005).

[16] L. Lok, "The need for speed in stochastic simulation," *Nature Biotechnology*, vol. 22, pp. 964-965, 2004.

[17] J. McCollum, J. Lancaster, D. W. Bouldin and G. D. Peterson, "Hardware Acceleration of Pseudo-Random Number Generation for Simulation Applications," *Proceedings, Southeastern Symposium on System Theory*, 2003.

[18] J. McCollum (2004). *Accelerating Exact Stochastic Simulation*. M.S. Thesis, University of Tennessee, Knoxville.

[19] S. Merchant, S. Fields, *Amirix AP130 VirtexIIPro System Tutorial (v1.0)*, Department of Electrical and Computer Engineering, The University of Tennessee, Knoxville, Marth 2005.

[20] J. McCollum, C. Cox, G. Peterson, M. Simpson and N. Samatova, "The Sorting Direct Method: An Efficient Stochastic Simulation Algorithm for Modeling Biochemical Systems," *The Journal Computational Biology and Chemistry*, 2005.

APPENDICES

Appendix A

Register Based Design VHDL

parith.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY parith IS
  PORT (    clk   : IN STD_LOGIC;
            we    : OUT STD_LOGIC;
            addr  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            din   : OUT STD_LOGIC_VECTOR(63 DOWNTO 0);
            dout  : IN STD_LOGIC_VECTOR(63 DOWNTO 0));
END parith;

ARCHITECTURE rtl OF parith IS

COMPONENT prop_1
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(20 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(27 DOWNTO 0) );
END COMPONENT;

COMPONENT prop_1_onoff
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(20 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
END COMPONENT;

COMPONENT prop_2
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
```

```
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(39 DOWNTO 0) );
END COMPONENT;

COMPONENT prop_2_onoff
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(27 DOWNTO 0) );
END COMPONENT;

COMPONENT prop_self
  PORT (    clk         : IN STD_LOGIC;
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(20 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(39 DOWNTO 0) );
```

```
END COMPONENT;

COMPONENT sumprop
  PORT (    clk   : IN STD_LOGIC;
            p0    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p1    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p2    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p3    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p4    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p5    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p6    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p7    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p8    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p9    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p10   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p11   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p12   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p13   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p14   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p15   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p16   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p17   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p18   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p19   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p20   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p21   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            totalp      : OUT STD_LOGIC_VECTOR(39 DOWNTO 0) );
END COMPONENT;

COMPONENT lfsr32
  PORT (    in_clock    : IN STD_LOGIC;
            in_reset    : IN STD_LOGIC;
            in_seed           : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            out_random_number : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END COMPONENT;

COMPONENT rxselect
  PORT (    clk         : IN STD_LOGIC;
            p0          : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p1          : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p2          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p3          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p4          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p5          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p6          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p7          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p8          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p9          : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p10         : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p11         : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p12         : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p13         : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p14         : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p15         : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p16         : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p17         : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p18         : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
```

```
            p19         : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p20         : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p21         : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            product     : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            selection   : OUT STD_LOGIC_VECTOR(4 DOWNTO 0) );
END COMPONENT;

COMPONENT updatespecies
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction0   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction1   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction2   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction3   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction4   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction5   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction6   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction7   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction8   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction9   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction10  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction11  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction12  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction13  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction14  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction15  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction16  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction17  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction18  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction19  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction20  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction21  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            selection   : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            newspecies0 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies1 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies2 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies3 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies4 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies5 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies6 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies7 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies8 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
```

```
            newspecies9 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies10      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies11      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies12      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies13      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies14      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies15      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0) );
END COMPONENT;

SIGNAL s_sp0           : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_sp1           : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_sp2           : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_sp3           : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_sp4           : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp5           : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp6           : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp7           : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp8           : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp9           : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp10          : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp11          : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp12          : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp13          : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp14          : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_sp15          : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_rx0           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx1           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx2           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx3           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx4           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx5           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx6           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx7           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx8           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx9           : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx10          : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx11          : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx12          : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_rx13          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx14          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx15          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx16          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx17          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx18          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx19          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx20          : STD_LOGIC_VECTOR(35 DOWNTO 0);
SIGNAL s_rx21          : STD_LOGIC_VECTOR(30 DOWNTO 0);
SIGNAL s_prop0         : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL s_prop1         : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL s_prop2         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop3         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop4         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop5         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop6         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop7         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop8         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop9         : STD_LOGIC_VECTOR(27 DOWNTO 0);
```

```
SIGNAL s_prop10         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop11         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop12         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop13         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop14         : STD_LOGIC_VECTOR(27 DOWNTO 0);
SIGNAL s_prop15         : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_prop16         : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_prop17         : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_prop18         : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_prop19         : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_prop20         : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_prop21         : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_tprop          : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL s_lfsr_enable    : STD_LOGIC;
SIGNAL s_lfsr_reset     : STD_LOGIC;
SIGNAL s_seed           : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL s_URV            : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL s_product  : STD_LOGIC_VECTOR(71 DOWNTO 0);
SIGNAL s_rxselect : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL s_newsp0         : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_newsp1         : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_newsp2         : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_newsp3         : STD_LOGIC_VECTOR(0 DOWNTO 0);
SIGNAL s_newsp4         : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp5         : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp6         : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp7         : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp8         : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp9         : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp10  : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp11  : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp12  : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp13  : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp14  : STD_LOGIC_VECTOR(11 DOWNTO 0);
SIGNAL s_newsp15  : STD_LOGIC_VECTOR(11 DOWNTO 0);

BEGIN

  m0 : prop_1_onoff PORT MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_rx0(20
DOWNTO 0),s_prop0);
  m1 : prop_1_onoff PORT MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_rx1(20
DOWNTO 0),s_prop1);
  m2 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx2(20 DOWNTO 0),s_prop2);
  m3 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx3(20 DOWNTO 0),s_prop3);
  m4 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx4(20 DOWNTO 0),s_prop4);
  m5 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx5(20 DOWNTO 0),s_prop5);
  m6 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx6(20 DOWNTO 0),s_prop6);
```

```
  m7 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx7(20 DOWNTO 0),s_prop7);
  m8 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx8(20 DOWNTO 0),s_prop8);
  m9 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx9(20 DOWNTO 0),s_prop9);
  m10 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx10(20 DOWNTO 0),s_prop10);
  m11 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx11(20 DOWNTO 0),s_prop11);
  m12 : prop_1 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx12(20 DOWNTO 0),s_prop12);
  m13 : prop_2_onoff PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx13(25 DOWNTO 0),s_prop13);
  m14 : prop_2_onoff PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx14(25 DOWNTO 0),s_prop14);
  m15 : prop_2 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx15(25 DOWNTO 0),s_prop15);
  m16 : prop_2 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx16(25 DOWNTO 0),s_prop16);
  m17 : prop_2 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx17(25 DOWNTO 0),s_prop17);
  m18 : prop_2 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx18(25 DOWNTO 0),s_prop18);
  m19 : prop_2 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx19(25 DOWNTO 0),s_prop19);
  m20 : prop_2 PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx20(25 DOWNTO 0),s_prop20);
  m21 : prop_self PORT
MAP(clk,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_sp10,s_sp11,s_sp12,s_sp13
,s_sp14,s_sp15,s_rx21(20 DOWNTO 0),s_prop21);
  m22 : sumprop PORT
MAP(clk,s_prop0,s_prop1,s_prop2,s_prop3,s_prop4,s_prop5,s_prop6,s_prop7
,s_prop8,s_prop9,s_prop10,s_prop11,s_prop12,s_prop13,s_prop14,s_prop15,
s_prop16,s_prop17,s_prop18,s_prop19,s_prop20,s_prop21,s_tprop);
  m23 : lfsr32 PORT MAP(s_lfsr_enable,s_lfsr_reset,s_seed,s_URV);
  m24 : rxselect PORT
MAP(clk,s_prop0,s_prop1,s_prop2,s_prop3,s_prop4,s_prop5,s_prop6,s_prop7
,s_prop8,s_prop9,s_prop10,s_prop11,s_prop12,s_prop13,s_prop14,s_prop15,
s_prop16,s_prop17,s_prop18,s_prop19,s_prop20,s_prop21,s_product(71
DOWNTO 32),s_rxselect);
  m25 : updatespecies PORT
MAP(clk,s_sp0,s_sp1,s_sp2,s_sp3,s_sp4,s_sp5,s_sp6,s_sp7,s_sp8,s_sp9,s_s
```

```
p10,s_sp11,s_sp12,s_sp13,s_sp14,s_sp15,s_rx0(30 DOWNTO 16),s_rx1(30
DOWNTO 16),s_rx2(30 DOWNTO 16),s_rx3(30 DOWNTO 16),s_rx4(30 DOWNTO
16),s_rx5(30 DOWNTO 16),s_rx6(30 DOWNTO 16),s_rx7(30 DOWNTO
16),s_rx8(30 DOWNTO 16),s_rx9(30 DOWNTO 16),s_rx10(30 DOWNTO
16),s_rx11(30 DOWNTO 16),s_rx12(30 DOWNTO 16),s_rx13(35 DOWNTO
16),s_rx14(35 DOWNTO 16),s_rx15(35 DOWNTO 16),s_rx16(35 DOWNTO
16),s_rx17(35 DOWNTO 16),s_rx18(35 DOWNTO 16),s_rx19(35 DOWNTO
16),s_rx20(35 DOWNTO 16),s_rx21(30 DOWNTO
16),s_rxselect,s_newsp0,s_newsp1,s_newsp2,s_newsp3,s_newsp4,s_newsp5,s_
newsp6,s_newsp7,s_newsp8,s_newsp9,s_newsp10,s_newsp11,s_newsp12,s_newsp
13,s_newsp14,s_newsp15);

  PROCESS (clk)
      VARIABLE species0 : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE species1 : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE species2 : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE species3 : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE species4 : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species5 : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species6 : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species7 : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species8 : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species9 : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species10      : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species11      : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species12      : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species13      : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species14      : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE species15      : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE reaction0      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction1      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction2      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction3      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction4      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction5      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction6      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction7      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction8      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction9      : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction10     : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction11     : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction12     : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE reaction13     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction14     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction15     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction16     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction17     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction18     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction19     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction20     : STD_LOGIC_VECTOR(35 DOWNTO 0);
      VARIABLE reaction21     : STD_LOGIC_VECTOR(30 DOWNTO 0);
      VARIABLE state          : STD_LOGIC_VECTOR(3 DOWNTO 0);
      VARIABLE state2         : STD_LOGIC_VECTOR(7 DOWNTO 0);
      VARIABLE product  : STD_LOGIC_VECTOR(71 DOWNTO 0);
      VARIABLE index          : STD_LOGIC_VECTOR(7 DOWNTO 0);
      VARIABLE maxindex : STD_LOGIC_VECTOR(7 DOWNTO 0);
      VARIABLE looping  : STD_LOGIC;
```

```
BEGIN
    IF (clk = '1' AND clk'EVENT) THEN
          s_sp0 <= species0; s_sp1 <= species1;
          s_sp2 <= species2; s_sp3 <= species3;
          s_sp4 <= species4; s_sp5 <= species5;
          s_sp6 <= species6; s_sp7 <= species7;
          s_sp8 <= species8; s_sp9 <= species9;
          s_sp10 <= species10; s_sp11 <= species11;
          s_sp12 <= species12; s_sp13 <= species13;
          s_sp14 <= species14; s_sp15 <= species15;
          s_rx0 <= reaction0; s_rx1 <= reaction1;
          s_rx2 <= reaction2; s_rx3 <= reaction3;
          s_rx4 <= reaction4; s_rx5 <= reaction5;
          s_rx6 <= reaction6; s_rx7 <= reaction7;
          s_rx8 <= reaction8; s_rx9 <= reaction9;
          s_rx10 <= reaction10; s_rx11 <= reaction11;
          s_rx12 <= reaction12; s_rx13 <= reaction13;
          s_rx14 <= reaction14; s_rx15 <= reaction15;
          s_rx16 <= reaction16; s_rx17 <= reaction17;
          s_rx18 <= reaction18; s_rx19 <= reaction19;
          s_rx20 <= reaction20; s_rx21 <= reaction21;
          product := s_URV * s_tprop;

          -- SET ADDRESS FROM WHICH TO READ COMMAND
          IF (state = "0000") THEN
                state2 := "00000000";
                state := state + 1;
                we <= '0';
                addr <= X"00";
                din <= (others => '0');
                s_lfsr_reset <= '0';
                s_lfsr_enable <= '0';
                index := X"02";
                maxindex := X"FC";
                looping := '0';

          -- INTERPRET COMMANDS
          ELSIF (state = "0001") THEN

                -- LOOPING THROUGH 250 REACTIONS
                IF (looping = '1') THEN
                      IF (index < maxindex) THEN
                            IF (state2 = "00000000") THEN
                                  we <= '1';
                                  addr <= index;
                                  din(63 DOWNTO 32) <= s_tprop(31
DOWNTO 0);
                                  din(4 DOWNTO 0) <= s_rxselect;
                                  species0 := s_newsp0;
                                  species1 := s_newsp1;
                                  species2 := s_newsp2;
                                  species3 := s_newsp3;
                                  species4 := s_newsp4;
                                  species5 := s_newsp5;
                                  species6 := s_newsp6;
                                  species7 := s_newsp7;
```

```
                        species8 := s_newsp8;
                        species9 := s_newsp9;
                        species10 := s_newsp10;
                        species11 := s_newsp11;
                        species12 := s_newsp12;
                        species13 := s_newsp13;
                        species14 := s_newsp14;
                        species15 := s_newsp15;
                        s_lfsr_reset <= '0';
                        s_lfsr_enable <= '1';
                        state2 := state2 + 1;
                  ELSIF (state2 = "00000001") THEN
                        we <= '0';
                        s_lfsr_reset <= '0';
                        s_lfsr_enable <= '0';
                        state2 := state2 + 1;
                  ELSIF (state2 = "00000101") THEN
                        we <= '0';
                        index := index + 1;
                        state2 := "00000000";
                  ELSE
                        we <= '0';
                        state2 := state2 + 1;
                  END IF;
            ELSE
                  we <= '0';
                  addr <= X"00";
                  looping := '0';
                  state := state + 1;
            END IF;

      -- NO-OP
      ELSIF (dout = X"0000") THEN
            we <= '0';
            addr <= X"00";
            state := "0000";

      -- SETTING A SPECIES POPULATION
      ELSIF (dout(63 DOWNTO 60) = "0001") THEN
            we <= '0';
            addr <= X"00";
            IF (dout(59 DOWNTO 55) = "00000") THEN
                  species0 := dout(0 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "00001") THEN
                  species1 := dout(0 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "00010") THEN
                  species2 := dout(0 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "00011") THEN
                  species3 := dout(0 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "00100") THEN
                  species4 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "00101") THEN
                  species5 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "00110") THEN
                  species6 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "00111") THEN
                  species7 := dout(11 DOWNTO 0);
```

```
            ELSIF (dout(59 DOWNTO 55) = "01000") THEN
                  species8 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "01001") THEN
                  species9 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "01010") THEN
                  species10 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "01011") THEN
                  species11 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "01100") THEN
                  species12 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "01101") THEN
                  species13 := dout(11 DOWNTO 0);
            ELSIF (dout(59 DOWNTO 55) = "01110") THEN
                  species14 := dout(11 DOWNTO 0);
            ELSE
                  species15 := dout(11 DOWNTO 0);
            END IF;
            state := state + 1;

      -- READING A SPECIES POPULATION
      ELSIF (dout(63 DOWNTO 60) = "0010") THEN
            we <= '1';
            addr <= X"01";
            IF (dout(59 DOWNTO 55) = "00000") THEN
                  din(0 DOWNTO 0) <= species0;
            ELSIF (dout(59 DOWNTO 55) = "00001") THEN
                  din(0 DOWNTO 0) <= species1;
            ELSIF (dout(59 DOWNTO 55) = "00010") THEN
                  din(0 DOWNTO 0) <= species2;
            ELSIF (dout(59 DOWNTO 55) = "00011") THEN
                  din(0 DOWNTO 0) <= species3;
            ELSIF (dout(59 DOWNTO 55) = "00100") THEN
                  din(11 DOWNTO 0) <= species4;
            ELSIF (dout(59 DOWNTO 55) = "00101") THEN
                  din(11 DOWNTO 0) <= species5;
            ELSIF (dout(59 DOWNTO 55) = "00110") THEN
                  din(11 DOWNTO 0) <= species6;
            ELSIF (dout(59 DOWNTO 55) = "00111") THEN
                  din(11 DOWNTO 0) <= species7;
            ELSIF (dout(59 DOWNTO 55) = "01000") THEN
                  din(11 DOWNTO 0) <= species8;
            ELSIF (dout(59 DOWNTO 55) = "01001") THEN
                  din(11 DOWNTO 0) <= species9;
            ELSIF (dout(59 DOWNTO 55) = "01010") THEN
                  din(11 DOWNTO 0) <= species10;
            ELSIF (dout(59 DOWNTO 55) = "01011") THEN
                  din(11 DOWNTO 0) <= species11;
            ELSIF (dout(59 DOWNTO 55) = "01100") THEN
                  din(11 DOWNTO 0) <= species12;
            ELSIF (dout(59 DOWNTO 55) = "01101") THEN
                  din(11 DOWNTO 0) <= species13;
            ELSIF (dout(59 DOWNTO 55) = "01110") THEN
                  din(11 DOWNTO 0) <= species14;
            ELSE
                  din(11 DOWNTO 0) <= species15;
            END IF;
            state := state + 1;
```

```
                  -- SETTING A REACTION EQUATION
                  ELSIF (dout(63 DOWNTO 60) = "0011") THEN
                        we <= '0';
                        addr <= X"00";
                        IF (dout(59 DOWNTO 55) = "00000") THEN
                              reaction0(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "00001") THEN
                              reaction1(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "00010") THEN
                              reaction2(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "00011") THEN
                              reaction3(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "00100") THEN
                              reaction4(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "00101") THEN
                              reaction5(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "00110") THEN
                              reaction6(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "00111") THEN
                              reaction7(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "01000") THEN
                              reaction8(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "01001") THEN
                              reaction9(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "01010") THEN
                              reaction10(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "01011") THEN
                              reaction11(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "01100") THEN
                              reaction12(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        ELSIF (dout(59 DOWNTO 55) = "01101") THEN
                              reaction13(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction13(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSIF (dout(59 DOWNTO 55) = "01110") THEN
                              reaction14(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction14(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSIF (dout(59 DOWNTO 55) = "01111") THEN
                              reaction15(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction15(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSIF (dout(59 DOWNTO 55) = "10000") THEN
                              reaction16(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction16(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSIF (dout(59 DOWNTO 55) = "10001") THEN
```

```
                              reaction17(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction17(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSIF (dout(59 DOWNTO 55) = "10010") THEN
                              reaction18(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction18(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSIF (dout(59 DOWNTO 55) = "10011") THEN
                              reaction19(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction19(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSIF (dout(59 DOWNTO 55) = "10100") THEN
                              reaction20(35 DOWNTO 26) := dout(41
DOWNTO 32); reaction20(25 DOWNTO 0) := dout(25 DOWNTO 0);
                        ELSE
                              reaction21(30 DOWNTO 0) := dout(30 DOWNTO
0);
                        END IF;
                        state := state + 1;

                  -- READING A REACTION EQUATION
      --          ELSIF (dout(63 DOWNTO 60) = "0100") THEN
      --                we <= '1';
      --                addr <= X"01";
      --                IF (dout(59 DOWNTO 55) = "00000") THEN
      --                      din(30 DOWNTO 0) <= reaction0(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "00001") THEN
      --                      din(30 DOWNTO 0) <= reaction1(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "00010") THEN
      --                      din(30 DOWNTO 0) <= reaction2(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "00011") THEN
      --                      din(30 DOWNTO 0) <= reaction3(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "00100") THEN
      --                      din(30 DOWNTO 0) <= reaction4(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "00101") THEN
      --                      din(30 DOWNTO 0) <= reaction5(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "00110") THEN
      --                      din(30 DOWNTO 0) <= reaction6(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "00111") THEN
      --                      din(30 DOWNTO 0) <= reaction7(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "01000") THEN
      --                      din(30 DOWNTO 0) <= reaction8(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "01001") THEN
      --                      din(30 DOWNTO 0) <= reaction9(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "01010") THEN
      --                      din(30 DOWNTO 0) <= reaction10(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "01011") THEN
      --                      din(30 DOWNTO 0) <= reaction11(30 DOWNTO
0);
```

```
      --                ELSIF (dout(59 DOWNTO 55) = "01100") THEN
      --                      din(30 DOWNTO 0) <= reaction12(30 DOWNTO
0);
      --                ELSIF (dout(59 DOWNTO 55) = "01101") THEN
      --                      din(41 DOWNTO 32) <= reaction13(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction13(25 DOWNTO 0);
      --                ELSIF (dout(59 DOWNTO 55) = "01110") THEN
      --                      din(41 DOWNTO 32) <= reaction14(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction14(25 DOWNTO 0);
      --                ELSIF (dout(59 DOWNTO 55) = "01111") THEN
      --                      din(41 DOWNTO 32) <= reaction15(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction15(25 DOWNTO 0);
      --                ELSIF (dout(59 DOWNTO 55) = "10000") THEN
      --                      din(41 DOWNTO 32) <= reaction16(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction16(25 DOWNTO 0);
      --                ELSIF (dout(59 DOWNTO 55) = "10001") THEN
      --                      din(41 DOWNTO 32) <= reaction17(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction17(25 DOWNTO 0);
      --                ELSIF (dout(59 DOWNTO 55) = "10010") THEN
      --                      din(41 DOWNTO 32) <= reaction18(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction18(25 DOWNTO 0);
      --                ELSIF (dout(59 DOWNTO 55) = "10011") THEN
      --                      din(41 DOWNTO 32) <= reaction19(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction19(25 DOWNTO 0);
      --                ELSIF (dout(59 DOWNTO 55) = "10100") THEN
      --                      din(41 DOWNTO 32) <= reaction20(35 DOWNTO
25); din(25 DOWNTO 0) <= reaction20(25 DOWNTO 0);
      --                ELSE
      --                      din(30 DOWNTO 0) <= reaction21(30 DOWNTO
0);
      --                END IF;
      --                state := state + 1;

      --          -- READING A PROPENSITY
      --          ELSIF (dout(63 DOWNTO 60) = "0110") THEN
      --                we <= '1';
      --                addr <= X"01";
      --                IF (dout(59 DOWNTO 55) = "00000") THEN
      --                      din(15 DOWNTO 0) <= s_prop0;
      --                ELSIF (dout(59 DOWNTO 55) = "00001") THEN
      --                      din(15 DOWNTO 0) <= s_prop1;
      --                ELSIF (dout(59 DOWNTO 55) = "00010") THEN
      --                      din(27 DOWNTO 0) <= s_prop2;
      --                ELSIF (dout(59 DOWNTO 55) = "00011") THEN
      --                      din(27 DOWNTO 0) <= s_prop3;
      --                ELSIF (dout(59 DOWNTO 55) = "00100") THEN
      --                      din(27 DOWNTO 0) <= s_prop4;
      --                ELSIF (dout(59 DOWNTO 55) = "00101") THEN
      --                      din(27 DOWNTO 0) <= s_prop5;
      --                ELSIF (dout(59 DOWNTO 55) = "00110") THEN
      --                      din(27 DOWNTO 0) <= s_prop6;
      --                ELSIF (dout(59 DOWNTO 55) = "00111") THEN
      --                      din(27 DOWNTO 0) <= s_prop7;
      --                ELSIF (dout(59 DOWNTO 55) = "01000") THEN
      --                      din(27 DOWNTO 0) <= s_prop8;
      --                ELSIF (dout(59 DOWNTO 55) = "01001") THEN
      --                      din(27 DOWNTO 0) <= s_prop9;
```

```
--                ELSIF (dout(59 DOWNTO 55) = "01010") THEN
--                      din(27 DOWNTO 0) <= s_prop10;
--                ELSIF (dout(59 DOWNTO 55) = "01011") THEN
--                      din(27 DOWNTO 0) <= s_prop11;
--                ELSIF (dout(59 DOWNTO 55) = "01100") THEN
--                      din(27 DOWNTO 0) <= s_prop12;
--                ELSIF (dout(59 DOWNTO 55) = "01101") THEN
--                      din(27 DOWNTO 0) <= s_prop13;
--                ELSIF (dout(59 DOWNTO 55) = "01110") THEN
--                      din(39 DOWNTO 0) <= s_prop14;
--                ELSIF (dout(59 DOWNTO 55) = "01111") THEN
--                      din(39 DOWNTO 0) <= s_prop15;
--                ELSIF (dout(59 DOWNTO 55) = "10000") THEN
--                      din(39 DOWNTO 0) <= s_prop16;
--                ELSIF (dout(59 DOWNTO 55) = "10001") THEN
--                      din(39 DOWNTO 0) <= s_prop17;
--                ELSIF (dout(59 DOWNTO 55) = "10010") THEN
--                      din(39 DOWNTO 0) <= s_prop18;
--                ELSIF (dout(59 DOWNTO 55) = "10011") THEN
--                      din(39 DOWNTO 0) <= s_prop19;
--                ELSIF (dout(59 DOWNTO 55) = "10100") THEN
--                      din(39 DOWNTO 0) <= s_prop20;
--                ELSE
--                      din(39 DOWNTO 0) <= s_prop21;
--                END IF;
--                state := state + 1;

            -- READING THE SUM OF ALL PROPENSITIES
--          ELSIF (dout(63 DOWNTO 60) = "0111") THEN
--                we <= '1';
--                addr <= X"01";
--                din(39 DOWNTO 0) <= s_tprop;
--                state := state + 1;

            -- SET SEED TO UNIFORM RANDOM NUMBER GENERATOR
            ELSIF (dout(63 DOWNTO 60) = "1000") THEN
                  we <= '0';
                  addr <= X"00";
                  IF (state2 = "00000000") THEN
                        s_seed <= dout(31 DOWNTO 0);
                        s_lfsr_reset <= '1';
                        state2 := state2 + 1;
                  ELSE
                        s_seed <= dout(31 DOWNTO 0);
                        s_lfsr_reset <= '1';
                        s_lfsr_enable <= '1';
                        state := state + 1;
                        state2 := "00000000";
                  END IF;

            -- READING UNIFORM RANDOM NUMBER
--          ELSIF (dout(63 DOWNTO 60) = "1001") THEN
--                we <= '1';
--                addr <= X"01";
--                din(31 DOWNTO 0) <= s_URV;
--                state := state + 1;
```

```
                -- CALCULATE A UNIFORM RANDOM NUMBER
        --          ELSIF (dout(63 DOWNTO 60) = "1010") THEN
        --                we <= '0';
        --                addr <= X"00";
        --                s_lfsr_reset <= '0';
        --                s_lfsr_enable <= '1';
        --                state := state + 1;

                -- READING PRODUCT OF TOTAL PROPENSITY * UNIFORM
RANDOM NUMBER
        --          ELSIF (dout(63 DOWNTO 60) = "1011") THEN
        --                we <= '1';
        --                addr <= X"01";
        --                din(31 DOWNTO 0) <= s_product(71 DOWNTO 40);
        --                state := state + 1;

                -- READING THE REACTION THAT WAS SELECTED
        --          ELSIF (dout(63 DOWNTO 60) = "1100") THEN
        --                we <= '1';
        --                addr <= X"01";
        --                din(4 DOWNTO 0) <= s_rxselect;
        --                state := state + 1;

                -- UPDATE THE SPECIES POPULATIONS
        --          ELSIF (dout(63 DOWNTO 60) = "1101") THEN
        --                we <= '0';
        --                addr <= X"00";
        --                species0 := s_newsp0;
        --                species1 := s_newsp1;
        --                species2 := s_newsp2;
        --                species3 := s_newsp3;
        --                species4 := s_newsp4;
        --                species5 := s_newsp5;
        --                species6 := s_newsp6;
        --                species7 := s_newsp7;
        --                species8 := s_newsp8;
        --                species9 := s_newsp9;
        --                species10 := s_newsp10;
        --                species11 := s_newsp11;
        --                species12 := s_newsp12;
        --                species13 := s_newsp13;
        --                species14 := s_newsp14;
        --                species15 := s_newsp15;
        --                state := state + 1;

                -- STEP THROUGH 250 REACTIONS
                ELSIF (dout(63 DOWNTO 60) = "1110") THEN
                      we <= '0';
                      addr <= X"00";
                      index := X"02";
                      maxindex := dout(7 DOWNTO 0);
                      looping := '1';
                      state2 := "00000000";
                END IF;

            -- TELL CPU THAT FPGA IS DONE
            ELSIF (state = "0010") THEN
```

```
                we <= '1';
                addr <= X"00";
                din <= (others => '0');
                state := "0000";
            ELSE
                we <= '0';
                addr <= X"00";
                state := state + 1;
            END IF;
            s_product <= product;
      END IF;
  END PROCESS;
END rtl;
```

prop_1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY prop_1 IS
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(20 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(27 DOWNTO 0) );
END prop_1;

ARCHITECTURE rtl OF prop_1 IS

BEGIN

  PROCESS(clk)
      VARIABLE Y : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE prop : STD_LOGIC_VECTOR(27 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            IF (reaction(20) = '1') THEN
                  prop(27 DOWNTO 0) := X"0000000";
            ELSIF (reaction(19 DOWNTO 16) < X"4") THEN
                  IF (reaction(19 DOWNTO 16) = X"0") THEN
                        Y(0 DOWNTO 0) := species0;
                  ELSIF (reaction(19 DOWNTO 16) = X"1") THEN
                        Y(0 DOWNTO 0) := species1;
                  ELSIF (reaction(19 DOWNTO 16) = X"2") THEN
                        Y(0 DOWNTO 0) := species2;
                  ELSIF (reaction(19 DOWNTO 16) = X"3") THEN
                        Y(0 DOWNTO 0) := species3;
                  END IF;
                  IF (Y(0) = '0') THEN
                        prop := X"0000000";
                  ELSE
                        prop(27 DOWNTO 16) := X"000";
                        prop(15 DOWNTO 0) := reaction(15 DOWNTO 0);
                  END IF;
```

```
            ELSE
                  IF (reaction(19 DOWNTO 16) = X"4") THEN
                        Y := species4;
                  ELSIF (reaction(19 DOWNTO 16) = X"5") THEN
                        Y := species5;
                  ELSIF (reaction(19 DOWNTO 16) = X"6") THEN
                        Y := species6;
                  ELSIF (reaction(19 DOWNTO 16) = X"7") THEN
                        Y := species7;
                  ELSIF (reaction(19 DOWNTO 16) = X"8") THEN
                        Y := species8;
                  ELSIF (reaction(19 DOWNTO 16) = X"9") THEN
                        Y := species9;
                  ELSIF (reaction(19 DOWNTO 16) = X"A") THEN
                        Y := species10;
                  ELSIF (reaction(19 DOWNTO 16) = X"B") THEN
                        Y := species11;
                  ELSIF (reaction(19 DOWNTO 16) = X"C") THEN
                        Y := species12;
                  ELSIF (reaction(19 DOWNTO 16) = X"D") THEN
                        Y := species13;
                  ELSIF (reaction(19 DOWNTO 16) = X"E") THEN
                        Y := species14;
                  ELSIF (reaction(19 DOWNTO 16) = X"F") THEN
                        Y := species15;
                  END IF;

                  prop := reaction(15 DOWNTO 0) * Y;
            END IF;

            propensity <= prop;
      END IF;
  END PROCESS;
END rtl;
```

prop_1_onoff.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY prop_1_onoff IS
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(20 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
END prop_1_onoff;

ARCHITECTURE rtl OF prop_1_onoff IS

BEGIN

  PROCESS(clk)
      VARIABLE X : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE prop : STD_LOGIC_VECTOR(15 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            IF (reaction(20) = '1') THEN
                  prop(15 DOWNTO 0) := X"0000";
            ELSE
                  IF (reaction(19 DOWNTO 16) = X"0") THEN
                        X := species0;
                  ELSIF (reaction(19 DOWNTO 16) = X"1") THEN
                        X := species1;
                  ELSIF (reaction(19 DOWNTO 16) = X"2") THEN
                        X := species2;
                  ELSIF (reaction(19 DOWNTO 16) = X"3") THEN
                        X := species3;
                  END IF;
                  IF (X(0) = '1') THEN
                        prop := reaction(15 DOWNTO 0);
                  ELSE
                        prop(15 DOWNTO 0) := X"0000";
                  END IF;
            END IF;

            propensity <= prop;
      END IF;
  END PROCESS;
END rtl;
```

prop_2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY prop_2 IS
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(39 DOWNTO 0) );
END prop_2;

ARCHITECTURE rtl OF prop_2 IS

BEGIN

  PROCESS(clk)
      VARIABLE REACTANT1 : STD_LOGIC_VECTOR(1 DOWNTO 0);
      VARIABLE X,Y : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE prop : STD_LOGIC_VECTOR(39 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            IF (reaction(25) = '1') THEN
                  REACTANT1 := "00";
            ELSIF (reaction(24 DOWNTO 21) < X"4") THEN
                  REACTANT1 := "01";
                  IF (reaction(24 DOWNTO 21) = X"0") THEN
                        X(0 DOWNTO 0) := species0;
                  ELSIF (reaction(24 DOWNTO 21) = X"1") THEN
                        X(0 DOWNTO 0) := species1;
                  ELSIF (reaction(24 DOWNTO 21) = X"2") THEN
                        X(0 DOWNTO 0) := species2;
                  ELSIF (reaction(24 DOWNTO 21) = X"3") THEN
                        X(0 DOWNTO 0) := species3;
                  END IF;
            ELSE
                  REACTANT1 := "10";
                  IF (reaction(24 DOWNTO 21) = X"4") THEN
                        X := species4;
```

```
                ELSIF (reaction(24 DOWNTO 21) = X"5") THEN
                        X := species5;
                ELSIF (reaction(24 DOWNTO 21) = X"6") THEN
                        X := species6;
                ELSIF (reaction(24 DOWNTO 21) = X"7") THEN
                        X := species7;
                ELSIF (reaction(24 DOWNTO 21) = X"8") THEN
                        X := species8;
                ELSIF (reaction(24 DOWNTO 21) = X"9") THEN
                        X := species9;
                ELSIF (reaction(24 DOWNTO 21) = X"A") THEN
                        X := species10;
                ELSIF (reaction(24 DOWNTO 21) = X"B") THEN
                        X := species11;
                ELSIF (reaction(24 DOWNTO 21) = X"C") THEN
                        X := species12;
                ELSIF (reaction(24 DOWNTO 21) = X"D") THEN
                        X := species13;
                ELSIF (reaction(24 DOWNTO 21) = X"E") THEN
                        X := species14;
                ELSIF (reaction(24 DOWNTO 21) = X"F") THEN
                        X := species15;
                END IF;
        END IF;

        IF (reaction(20) = '1') THEN
                IF (REACTANT1 = "00") THEN
                        prop := X"0000000000";
                ELSIF (REACTANT1 = "01") THEN
                        IF (X(0) = '1') THEN
                                prop(39 DOWNTO 16) := X"000000";
                                prop(15 DOWNTO 0) := reaction(15 DOWNTO
0);
                        ELSE
                                prop := X"0000000000";
                        END IF;
                ELSE
                        prop(39 DOWNTO 28) := X"000";
                        prop(27 DOWNTO 0) := reaction(15 DOWNTO 0) * X;
                END IF;
        ELSIF (reaction(25 DOWNTO 21) = reaction(20 DOWNTO 16))
THEN
                Y := X - 1;
                prop := reaction(15 DOWNTO 0) * X * Y;
                prop(38 DOWNTO 0) := prop(39 DOWNTO 1);
                prop(39) := '0';
        ELSE
                IF (reaction(20 DOWNTO 16) = X"4") THEN
                        Y := species4;
                ELSIF (reaction(20 DOWNTO 16) = X"5") THEN
                        Y := species5;
                ELSIF (reaction(20 DOWNTO 16) = X"6") THEN
                        Y := species6;
                ELSIF (reaction(20 DOWNTO 16) = X"7") THEN
                        Y := species7;
                ELSIF (reaction(20 DOWNTO 16) = X"8") THEN
                        Y := species8;
```

```
                    ELSIF (reaction(20 DOWNTO 16) = X"9") THEN
                          Y := species9;
                    ELSIF (reaction(20 DOWNTO 16) = X"A") THEN
                          Y := species10;
                    ELSIF (reaction(20 DOWNTO 16) = X"B") THEN
                          Y := species11;
                    ELSIF (reaction(20 DOWNTO 16) = X"C") THEN
                          Y := species12;
                    ELSIF (reaction(20 DOWNTO 16) = X"D") THEN
                          Y := species13;
                    ELSIF (reaction(20 DOWNTO 16) = X"E") THEN
                          Y := species14;
                    ELSIF (reaction(20 DOWNTO 16) = X"F") THEN
                          Y := species15;
                    END IF;
                    IF (REACTANT1 = "00") THEN
                          prop(39 DOWNTO 28) := X"000";
                          prop(27 DOWNTO 0) := reaction(15 DOWNTO 0) * Y;
                    ELSIF (REACTANT1 = "01") THEN
                          IF (X(0) = '1') THEN
                                prop(39 DOWNTO 28) := X"000";
                                prop(27 DOWNTO 0) := reaction(15 DOWNTO
0) * Y;
                          ELSE
                                prop := X"0000000000";
                          END IF;
                    ELSE
                          prop := reaction(15 DOWNTO 0) * X * Y;
                    END IF;
              END IF;

              propensity <= prop;
        END IF;
    END PROCESS;
END rtl;
```

prop_2_onoff.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY prop_2_onoff IS
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(27 DOWNTO 0) );
END prop_2_onoff;

ARCHITECTURE rtl OF prop_2_onoff IS

BEGIN

  PROCESS(clk)
      VARIABLE X : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE Y : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE prop : STD_LOGIC_VECTOR(27 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            IF (reaction(20) = '1') THEN
                  IF (reaction(25) = '1') THEN
                        prop(27 DOWNTO 0) := X"0000000";
                  ELSE
                        IF (reaction(24 DOWNTO 21) = X"0") THEN
                              X(0 DOWNTO 0) := species0;
                        ELSIF (reaction(24 DOWNTO 21) = X"1") THEN
                              X(0 DOWNTO 0) := species1;
                        ELSIF (reaction(24 DOWNTO 21) = X"2") THEN
                              X(0 DOWNTO 0) := species2;
                        ELSIF (reaction(24 DOWNTO 21) = X"3") THEN
                              X(0 DOWNTO 0) := species3;
                        END IF;
                        IF (X(0) = '0') THEN
                              prop(27 DOWNTO 0) := X"0000000";
                        ELSE
                              prop(27 DOWNTO 16) := X"000";
```

```
                              prop(15 DOWNTO 0) := reaction(15 DOWNTO
0);
                        END IF;
                  END IF;
            ELSE
                  IF (reaction(19 DOWNTO 16) = X"4") THEN
                        Y := species4;
                  ELSIF (reaction(19 DOWNTO 16) = X"5") THEN
                        Y := species5;
                  ELSIF (reaction(19 DOWNTO 16) = X"6") THEN
                        Y := species6;
                  ELSIF (reaction(19 DOWNTO 16) = X"7") THEN
                        Y := species7;
                  ELSIF (reaction(19 DOWNTO 16) = X"8") THEN
                        Y := species8;
                  ELSIF (reaction(19 DOWNTO 16) = X"9") THEN
                        Y := species9;
                  ELSIF (reaction(19 DOWNTO 16) = X"A") THEN
                        Y := species10;
                  ELSIF (reaction(19 DOWNTO 16) = X"B") THEN
                        Y := species11;
                  ELSIF (reaction(19 DOWNTO 16) = X"C") THEN
                        Y := species12;
                  ELSIF (reaction(19 DOWNTO 16) = X"D") THEN
                        Y := species13;
                  ELSIF (reaction(19 DOWNTO 16) = X"E") THEN
                        Y := species14;
                  ELSIF (reaction(19 DOWNTO 16) = X"F") THEN
                        Y := species15;
                  END IF;
                  IF (reaction(25) = '1') THEN
                        prop := reaction(15 DOWNTO 0) * Y;
                  ELSE
                        IF (reaction(24 DOWNTO 21) = X"0") THEN
                              X(0 DOWNTO 0) := species0;
                        ELSIF (reaction(24 DOWNTO 21) = X"1") THEN
                              X(0 DOWNTO 0) := species1;
                        ELSIF (reaction(24 DOWNTO 21) = X"2") THEN
                              X(0 DOWNTO 0) := species2;
                        ELSIF (reaction(24 DOWNTO 21) = X"3") THEN
                              X(0 DOWNTO 0) := species3;
                        END IF;
                        IF (X(0) = '0') THEN
                              prop(27 DOWNTO 0) := X"0000000";
                        ELSE
                              prop := reaction(15 DOWNTO 0) * Y;
                        END IF;
                  END IF;
            END IF;

            propensity <= prop;
      END IF;
  END PROCESS;
END rtl;
```

prop_self.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY prop_self IS
  PORT (    clk         : IN STD_LOGIC;
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction    : IN STD_LOGIC_VECTOR(20 DOWNTO 0);
            propensity  : OUT STD_LOGIC_VECTOR(39 DOWNTO 0) );
END prop_self;

ARCHITECTURE rtl OF prop_self IS

BEGIN

  PROCESS(clk)
      VARIABLE X, Y : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE prop : STD_LOGIC_VECTOR(39 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            IF (reaction(20) = '1') THEN
                  prop := X"0000000000";
            ELSE
                  IF (reaction(19 DOWNTO 16) = X"4") THEN
                        X := species4;
                  ELSIF (reaction(19 DOWNTO 16) = X"5") THEN
                        X := species5;
                  ELSIF (reaction(19 DOWNTO 16) = X"6") THEN
                        X := species6;
                  ELSIF (reaction(19 DOWNTO 16) = X"7") THEN
                        X := species7;
                  ELSIF (reaction(19 DOWNTO 16) = X"8") THEN
                        X := species8;
                  ELSIF (reaction(19 DOWNTO 16) = X"9") THEN
                        X := species9;
                  ELSIF (reaction(19 DOWNTO 16) = X"A") THEN
                        X := species10;
                  ELSIF (reaction(19 DOWNTO 16) = X"B") THEN
                        X := species11;
                  ELSIF (reaction(19 DOWNTO 16) = X"C") THEN
                        X := species12;
                  ELSIF (reaction(19 DOWNTO 16) = X"D") THEN
```

```
                        X := species13;
                    ELSIF (reaction(19 DOWNTO 16) = X"E") THEN
                        X := species14;
                    ELSIF (reaction(19 DOWNTO 16) = X"F") THEN
                        X := species15;
                    END IF;

                    Y := X - 1;
                    prop := reaction(15 DOWNTO 0) * X * Y;
                    prop(38 DOWNTO 0) := prop(39 DOWNTO 1);
                    prop(39) := '0';

                END IF;

                propensity <= prop;
            END IF;
        END PROCESS;
    END rtl;
```

rxselect.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY rxselect IS
  PORT (    clk         : IN STD_LOGIC;
            p0    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p1    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p2    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p3    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p4    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p5    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p6    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p7    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p8    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p9    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p10   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p11   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p12   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p13   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p14   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p15   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p16   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p17   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p18   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p19   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p20   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p21   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            product           : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            selection   : OUT STD_LOGIC_VECTOR(4 DOWNTO 0) );
END rxselect;

ARCHITECTURE rtl OF rxselect IS

BEGIN

  PROCESS(clk)
      VARIABLE rxselect : STD_LOGIC_VECTOR(4 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            IF (product < p0) THEN
                  rxselect := "00000";
            ELSIF (product < (p0 + p1)) THEN
                  rxselect := "00001";
            ELSIF (product < (p0 + p1 + p2)) THEN
                  rxselect := "00010";
            ELSIF (product < (p0 + p1 + p2 + p3)) THEN
                  rxselect := "00011";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4)) THEN
                  rxselect := "00100";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5)) THEN
                  rxselect := "00101";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6)) THEN
```

```
                rxselect := "00110";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7))
THEN
                rxselect := "00111";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8)) THEN
                rxselect := "01000";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9)) THEN
                rxselect := "01001";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10)) THEN
                rxselect := "01010";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11)) THEN
                rxselect := "01011";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12)) THEN
                rxselect := "01100";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13)) THEN
                rxselect := "01101";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13 + p14)) THEN
                rxselect := "01110";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13 + p14 + p15)) THEN
                rxselect := "01111";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13 + p14 + p15 + p16)) THEN
                rxselect := "10000";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13 + p14 + p15 + p16 + p17)) THEN
                rxselect := "10001";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13 + p14 + p15 + p16 + p17 + p18)) THEN
                rxselect := "10010";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13 + p14 + p15 + p16 + p17 + p18 + p19))
THEN
                rxselect := "10011";
            ELSIF (product < (p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 +
p8 + p9 + p10 + p11 + p12 + p13 + p14 + p15 + p16 + p17 + p18 + p19 +
p20)) THEN
                rxselect := "10100";
            ELSE
                rxselect := "10101";
            END IF;

            selection <= rxselect;
      END IF;
  END PROCESS;
END rtl;
```

sumprop.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY sumprop IS
  PORT (    clk   : IN STD_LOGIC;
            p0    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p1    : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            p2    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p3    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p4    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p5    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p6    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p7    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p8    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p9    : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p10   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p11   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p12   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p13   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p14   : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
            p15   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p16   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p17   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p18   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p19   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p20   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            p21   : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
            totalp      : OUT STD_LOGIC_VECTOR(39 DOWNTO 0) );
END sumprop;

ARCHITECTURE rtl OF sumprop IS

BEGIN

  PROCESS(clk)
      VARIABLE sum : STD_LOGIC_VECTOR(39 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            sum := p0 + p1 + p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9 +
p10 + p11 + p12 + p13 + p14 + p15 + p16 + p17 + p18 + p19 + p20 + p21;

            totalp <= sum;
      END IF;
  END PROCESS;
END rtl;
```

updatespecies.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY updatespecies IS
  PORT (    clk         : IN STD_LOGIC;
            species0    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species1    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species2    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species3    : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
            species4    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species5    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species6    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species7    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species8    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species9    : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species10   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species11   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species12   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species13   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species14   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            species15   : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
            reaction0   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction1   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction2   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction3   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction4   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction5   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction6   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction7   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction8   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction9   : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction10  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction11  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction12  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            reaction13  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction14  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction15  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction16  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction17  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction18  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction19  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction20  : IN STD_LOGIC_VECTOR(19 DOWNTO 0);
            reaction21  : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
            selection   : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            newspecies0 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies1 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies2 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies3 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
            newspecies4 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies5 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies6 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies7 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies8 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
```

```
            newspecies9 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies10      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies11      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies12      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies13      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies14      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
            newspecies15      : OUT STD_LOGIC_VECTOR(11 DOWNTO 0) );
END updatespecies;

ARCHITECTURE rtl OF updatespecies IS

BEGIN

  PROCESS (clk)
      VARIABLE rx       : STD_LOGIC_VECTOR(19 DOWNTO 0);
      VARIABLE newsp0         : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE newsp1         : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE newsp2         : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE newsp3         : STD_LOGIC_VECTOR(0 DOWNTO 0);
      VARIABLE newsp4         : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp5         : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp6         : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp7         : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp8         : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp9         : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp10  : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp11  : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp12  : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp13  : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp14  : STD_LOGIC_VECTOR(11 DOWNTO 0);
      VARIABLE newsp15  : STD_LOGIC_VECTOR(11 DOWNTO 0);

  BEGIN
      IF (clk = '1' AND clk'EVENT) THEN
            newsp0 := species0;
            newsp1 := species1;
            newsp2 := species2;
            newsp3 := species3;
            newsp4 := species4;
            newsp5 := species5;
            newsp6 := species6;
            newsp7 := species7;
            newsp8 := species8;
            newsp9 := species9;
            newsp10 := species10;
            newsp11 := species11;
            newsp12 := species12;
            newsp13 := species13;
            newsp14 := species14;
            newsp15 := species15;

            IF (selection = "00000") THEN
                  rx(4 DOWNTO 0) := "11111";
                  rx(19 DOWNTO 5) := reaction0;
            ELSIF (selection = "00001") THEN
                  rx(4 DOWNTO 0) := "11111";
                  rx(19 DOWNTO 5) := reaction1;
```

```
ELSIF (selection = "00010") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction2;
ELSIF (selection = "00011") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction3;
ELSIF (selection = "00100") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction4;
ELSIF (selection = "00101") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction5;
ELSIF (selection = "00110") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction6;
ELSIF (selection = "00111") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction7;
ELSIF (selection = "01000") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction8;
ELSIF (selection = "01001") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction9;
ELSIF (selection = "01010") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction10;
ELSIF (selection = "01011") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction11;
ELSIF (selection = "01100") THEN
      rx(4 DOWNTO 0) := "11111";
      rx(19 DOWNTO 5) := reaction12;
ELSIF (selection = "01101") THEN
      rx := reaction13;
ELSIF (selection = "01110") THEN
      rx := reaction14;
ELSIF (selection = "01111") THEN
      rx := reaction15;
ELSIF (selection = "10000") THEN
      rx := reaction16;
ELSIF (selection = "10001") THEN
      rx := reaction17;
ELSIF (selection = "10010") THEN
      rx := reaction18;
ELSIF (selection = "10011") THEN
      rx := reaction19;
ELSIF (selection = "10100") THEN
      rx := reaction20;
ELSE
      rx(4 DOWNTO 0) := reaction21(9 DOWNTO 5);
      rx(19 DOWNTO 5) := reaction21;
END IF;
IF (rx(9) /= '1') THEN
      CASE rx(8 DOWNTO 5) IS
            WHEN X"0" => newsp0 := newsp0 - 1;
            WHEN X"1" => newsp1 := newsp1 - 1;
```

```
            WHEN X"2" => newsp2 := newsp2 - 1;
            WHEN X"3" => newsp3 := newsp3 - 1;
            WHEN X"4" => newsp4 := newsp4 - 1;
            WHEN X"5" => newsp5 := newsp5 - 1;
            WHEN X"6" => newsp6 := newsp6 - 1;
            WHEN X"7" => newsp7 := newsp7 - 1;
            WHEN X"8" => newsp8 := newsp8 - 1;
            WHEN X"9" => newsp9 := newsp9 - 1;
            WHEN X"A" => newsp10 := newsp10 - 1;
            WHEN X"B" => newsp11 := newsp11 - 1;
            WHEN X"C" => newsp12 := newsp12 - 1;
            WHEN X"D" => newsp13 := newsp13 - 1;
            WHEN X"E" => newsp14 := newsp14 - 1;
            WHEN others => newsp15 := newsp15 - 1;
      END CASE;
END IF;
IF (rx(4) /= '1') THEN
      CASE rx(3 DOWNTO 0) IS
            WHEN X"0" => newsp0 := newsp0 - 1;
            WHEN X"1" => newsp1 := newsp1 - 1;
            WHEN X"2" => newsp2 := newsp2 - 1;
            WHEN X"3" => newsp3 := newsp3 - 1;
            WHEN X"4" => newsp4 := newsp4 - 1;
            WHEN X"5" => newsp5 := newsp5 - 1;
            WHEN X"6" => newsp6 := newsp6 - 1;
            WHEN X"7" => newsp7 := newsp7 - 1;
            WHEN X"8" => newsp8 := newsp8 - 1;
            WHEN X"9" => newsp9 := newsp9 - 1;
            WHEN X"A" => newsp10 := newsp10 - 1;
            WHEN X"B" => newsp11 := newsp11 - 1;
            WHEN X"C" => newsp12 := newsp12 - 1;
            WHEN X"D" => newsp13 := newsp13 - 1;
            WHEN X"E" => newsp14 := newsp14 - 1;
            WHEN others => newsp15 := newsp15 - 1;
      END CASE;
END IF;
IF (rx(19) /= '1') THEN
      CASE rx(18 DOWNTO 15) IS
            WHEN X"0" => newsp0 := newsp0 + 1;
            WHEN X"1" => newsp1 := newsp1 + 1;
            WHEN X"2" => newsp2 := newsp2 + 1;
            WHEN X"3" => newsp3 := newsp3 + 1;
            WHEN X"4" => newsp4 := newsp4 + 1;
            WHEN X"5" => newsp5 := newsp5 + 1;
            WHEN X"6" => newsp6 := newsp6 + 1;
            WHEN X"7" => newsp7 := newsp7 + 1;
            WHEN X"8" => newsp8 := newsp8 + 1;
            WHEN X"9" => newsp9 := newsp9 + 1;
            WHEN X"A" => newsp10 := newsp10 + 1;
            WHEN X"B" => newsp11 := newsp11 + 1;
            WHEN X"C" => newsp12 := newsp12 + 1;
            WHEN X"D" => newsp13 := newsp13 + 1;
            WHEN X"E" => newsp14 := newsp14 + 1;
            WHEN others => newsp15 := newsp15 + 1;
      END CASE;
END IF;
IF (rx(14) /= '1') THEN
```

```
                    CASE rx(13 DOWNTO 10) IS
                          WHEN X"0" => newsp0 := newsp0 + 1;
                          WHEN X"1" => newsp1 := newsp1 + 1;
                          WHEN X"2" => newsp2 := newsp2 + 1;
                          WHEN X"3" => newsp3 := newsp3 + 1;
                          WHEN X"4" => newsp4 := newsp4 + 1;
                          WHEN X"5" => newsp5 := newsp5 + 1;
                          WHEN X"6" => newsp6 := newsp6 + 1;
                          WHEN X"7" => newsp7 := newsp7 + 1;
                          WHEN X"8" => newsp8 := newsp8 + 1;
                          WHEN X"9" => newsp9 := newsp9 + 1;
                          WHEN X"A" => newsp10 := newsp10 + 1;
                          WHEN X"B" => newsp11 := newsp11 + 1;
                          WHEN X"C" => newsp12 := newsp12 + 1;
                          WHEN X"D" => newsp13 := newsp13 + 1;
                          WHEN X"E" => newsp14 := newsp14 + 1;
                          WHEN others => newsp15 := newsp15 + 1;
                    END CASE;
              END IF;

              newspecies0 <= newsp0;
              newspecies1 <= newsp1;
              newspecies2 <= newsp2;
              newspecies3 <= newsp3;
              newspecies4 <= newsp4;
              newspecies5 <= newsp5;
              newspecies6 <= newsp6;
              newspecies7 <= newsp7;
              newspecies8 <= newsp8;
              newspecies9 <= newsp9;
              newspecies10 <= newsp10;
              newspecies11 <= newsp11;
              newspecies12 <= newsp12;
              newspecies13 <= newsp13;
              newspecies14 <= newsp14;
              newspecies15 <= newsp15;
        END IF;
    END PROCESS;
END rtl;
```

lfsr32.vhd [17]

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity lfsr32 is
  port   ( in_clock          : in std_logic;
         in_reset          : in std_logic;
         in_seed           : in std_logic_vector(31 downto 0);
         out_random_number : out std_logic_vector(31 downto 0));
end entity lfsr32;

architecture a of lfsr32 is
begin
  process(in_clock)
    variable var_current_number : std_logic_vector(31 downto 0);
    variable var_startup : natural;
    variable var_next_bit : std_logic;
  begin
    if (in_clock = '1' and in_clock'event) then
      if (in_reset='1' or var_startup=0) then
      var_current_number := in_seed;
      var_startup := 1;
      else
      var_next_bit := var_current_number(0) XOR
                  var_current_number(26) XOR
                  var_current_number(27) XOR
                  var_current_number(31);
      var_current_number(31 downto 1) := var_current_number(30 downto
0);
      var_current_number(0) := var_next_bit;
      end if;
      out_random_number <= var_current_number;
    end if;
  end process;
end architecture a;
```

Appendix B

Register Based Design C++

hw.cc

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <math.h>
#include <iostream>
#include <cstdlib>
#include "iflib.h"

using namespace std;

#define NULLSPECIES 31
#define NMAX 16
#define MMAX 22
#define PMAX 4095
#define KMAX 65535

class CR{
  public:
      unsigned int reactants,products,fpk;
      double k;
      unsigned int *renum,*rewt,*prnum,*prwt;
};

char *memp;
int64 data;
int fd,tprop[250],rxselect[250];
unsigned int n,m,seed,*X,iterations,num,*mon,thecount;
CR *R;
double thetime,tau;
FILE *outFile;

void init(void) {
      fd = open(DEVICE, O_RDWR);
      memp = (char *)mmap(NULL, MTRRZ, PROT_READ, MAP_PRIVATE, fd, 0);
      if (memp  == MAP_FAILED) {
            perror(DEVICE);
            exit(1);
      }
      srand(time(NULL));
}

// Prints a number in Binary
```

```
void printBinary(unsigned int val,int index){
      int count;
      char chars[64];

      for(count=0;count<64;count++){
            chars[count]='0';
      }
      count = 0;
      do{
            if(val % 2 == 0) chars[count++] = '0';
            else chars[count++] = '1';
            val = val / 2;
      }while(val);
      count=index-1;
      while(count >= 0){
            if((count+1) % 4 == 0) printf(" ");
            printf("%c", chars[count--]);
      }
      printf("\n");
}

void setspeciespop(int index, int value){
      if(index < 4){
            if(value > 1) value = 1;
      }
      data.w[1] = (0x1<<28) + (index<<23);
      data.w[0] = value;
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }
}

int readspeciespop(int index){
      data.w[1] = (0x2<<28) + (index<<23);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      return (data.w[0] & 0xFFF);
}

void setreaction(int index, int reactant1, int reactant2, int product1,
int product2, int rate){
      if((index<13)||(index==21)){
            data.w[1] = (0x3<<28) + (index<<23);
            data.w[0] = (product1<<26) + (product2<<21) +
(reactant2<<16) + rate;
      }
      else{
```

```
            data.w[1] = (0x3<<28) + (index<<23) + (product1<<5) +
product2;
            data.w[0] = (reactant1<<21) + (reactant2<<16) + rate;
      }
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }
}

int readreaction(int index){
      data.w[1] = (0x4<<28) + (index<<24);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      return (((data.w[1]<<10) + (data.w[0]>>16)) & 0xFFFFF);
}

int readpropensity(int index){
      data.w[1] = (0x6<<28) + (index<<24);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      //return (((data.w[1]<<24) + data.w[0]>>8) & 0xFFFFFFFF);
      return data.w[0];
}

int readsum(void){
      data.w[1] = (0x7<<28);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      //return (((data.w[1]<<24) + data.w[0]>>8) & 0xFFFFFFFF);
      return data.w[0];
}

void setseed(int seed){
      data.w[1] = (0x8<<28);
      data.w[0] = seed;
      write64(data, memp+(0<<3));
```

```
	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}
}

unsigned int readURV(void){
	data.w[1] = (0x9<<28);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}

	read64(&data, memp+(1<<3));
	return (data.w[0]);
}

void nextURV(void){
	data.w[1] = (0xA<<28);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}
}

unsigned int readproduct(void){
	data.w[1] = (0xB<<28);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}

	read64(&data, memp+(1<<3));
	return (data.w[0]);
}

int readrxselected(void){
	data.w[1] = (0xC<<28);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}

	read64(&data, memp+(1<<3));
	return (data.w[0] & 0xF);
}

void updatespecies(void){
```

```
	data.w[1] = (0xD<<28);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}
}

void printresults(void){
	int i,j;

	for(i=0;i<250;i++){
		if(tprop[i]==-1) break;
		thetime+=(-
1/(double)tprop[i])*log((double)rand()/(double)RAND_MAX);
		// Update species populations
		for(j=0;j<R[rxselect[i]].reactants;j++){
			X[R[rxselect[i]].renum[j]]-=R[rxselect[i]].rewt[j];
		}
		for(j=0;j<R[rxselect[i]].products;j++){
			X[R[rxselect[i]].prnum[j]]+=R[rxselect[i]].prwt[j];
		}

		/*
		fprintf(outFile,"%6d %8.6lf",thecount,thetime);
		for(j=0;j<num;j++){
			fprintf(outFile," %4u",X[mon[j]]);
		}
		fprintf(outFile,"\n");
		thecount++;
		*/
	}
}


void step(int runs){
	int i,a=0;

	while(runs>0){
		// Tell FPGA to begin executing reactions
		data.w[1] = (0xE<<28);
		if(runs>=250) data.w[0] = 252;
		else data.w[0] = runs + 2;
		write64(data, memp+(0<<3));
		// Print previous results, on first pass there are no
previous results to print
		if(a==1) printresults();
		else a=1;
		// Wait until FPGA is done
		read64(&data, memp+(0<<3));
		while(data.w[1]!=0x0){
			read64(&data, memp+(0<<3));
		}
		// Update total propensity and reaction selected arrays
		//if(runs>=250){
			for(i=2;i<252;i++){
```

```
                        read64(&data, memp+(i<<3));
                        tprop[i-2] = data.w[1];
                        rxselect[i-2] = data.w[0];
                  }
            /*
            }
            else{
                  for(i=2;i<2+runs;i++){
                        read64(&data, memp+(i<<3));
                        tprop[i-2] = data.w[1];
                        rxselect[i-2] = data.w[0];
                  }
                  tprop[i] = -1;
            }
            */

            runs-=250;
      }
      printresults();
}

int main (int argc, char **argv)
{
      int
species[16],i,j,k,l,x,reaction[16],propensity[16],sum,selection,reacdat
a[4];
      unsigned int kl_int,MF=1,URV,product;
      double kl=1.0,y;
      char temp[51],c=65;
      FILE *inFile;
      struct timeval ts,te;

      outFile = fopen("results.txt","wt");

      if(argc>2){
            fprintf(stderr,"ERROR!  Expected usage: ./rchw [model
file]\n");
            exit(1);
      }
      if(argc==2) strcpy(temp,argv[1]);
      else{
            printf("Please enter the name of the model file to read
from: ");
            if(fgets(temp,50,stdin)==NULL){ printf("\n"); exit(0); }
            temp[strlen(temp)-1]='\0';
      }
      inFile = fopen(temp,"r");
      while(inFile == NULL){
            fprintf(stderr,"ERROR!  Unable to open: %s\n",temp);
            printf("Please enter the name of the model file to read
from: ");
            if(fgets(temp,50,stdin)==NULL){ printf("\n"); exit(0); }
            temp[strlen(temp)-1]='\0';
            inFile = fopen(temp,"r");
      }

      init();
```

```
	// Clear BRAM
	for(i=0;i<255;i++){
		data.w[1] = 0x0;
		data.w[0] = 0x0;
		write64(data, memp+(i<<3));
	}

	gettimeofday(&ts,NULL);

	// READ IN VARIABLES
	fscanf(inFile,"%u",&n);
	if(n>NMAX){
		fprintf(stderr,"ERROR!  Number of species exceeds limit of
%d\n",NMAX);
		exit(1);
	}
	X = new unsigned int[n];
	for(i=0;i<n;i++){
		fscanf(inFile,"%u",&X[i]);
		if((i<4)&&(X[i]>1)){
			X[i]=1;
			fprintf(stderr,"WARNING!  Species 0->3 are one bit,
so species %d has been set to 1\n",i);
		}
		if((i>3)&&(X[i]>PMAX)){
			X[i]=PMAX;
			fprintf(stderr,"WARNING!  Species 4->15 are twelve
bits, so species %d has been set to %d\n",i,PMAX);
		}
	}
	fscanf(inFile,"%u",&m);
	if(m>MMAX){
		fprintf(stderr,"ERROR!  Number of reactions exceeds limit
of %d\n",MMAX);
		exit(1);
	}
	R = new CR[m];
	for(i=0;i<m;i++){
		fscanf(inFile,"%d",&R[i].reactants);
		R[i].renum = new unsigned int[R[i].reactants];
		R[i].rewt = new unsigned int[R[i].reactants];
		k=0;
		for(j=0;j<R[i].reactants;j++){
			fscanf(inFile,"%u",&R[i].rewt[j]);
			k+=R[i].rewt[j];
			fscanf(inFile,"%u",&R[i].renum[j]);
		}
		if(k>2){
			fprintf(stderr,"ERROR!  Number of reactants in
reaction %d exceeds maximum of 2\n",i);
			exit(1);
		}
		if((i<4)&&(k>1)){
			fprintf(stderr,"ERROR!  Reactions 0->3 have a limit
of 1 reactant, reaction %d exceeds that\n",i);
			exit(1);
```

```
            }
            fscanf(inFile,"%d",&R[i].products);
            R[i].prnum = new unsigned int[R[i].products];
            R[i].prwt = new unsigned int[R[i].products];
            k=0;
            for(j=0;j<R[i].products;j++){
                  fscanf(inFile,"%u",&R[i].prwt[j]);
                  k+=R[i].rewt[j];
                  fscanf(inFile,"%u",&R[i].prnum[j]);
            }
            if(k>2){
                  fprintf(stderr,"ERROR!  Number of products in
reaction %d exceeds maximum of 2\n",i);
                  exit(1);
            }
            fscanf(inFile,"%lf",&R[i].k);
            y=R[i].k - (unsigned int)(R[i].k);
            if((y>0) && (y<kl)) kl=y;
      }
      if(fscanf(inFile,"%u",&num)==EOF){
            num=n;
            mon=new unsigned int[num];
            for(i=0;i<num;i++){ mon[i]=i; }
            seed=-1-(time(NULL));
            iterations=1000000;
      }
      else{
            mon=new unsigned int[num];
            for(i=0;i<num;i++){ fscanf(inFile,"%u",&mon[i]); }
            if(fscanf(inFile,"%u",&seed)==EOF){
                  seed=-1-(time(NULL));
                  iterations=1000000;
            }
            else{
                  if(fscanf(inFile,"%u",&iterations)==EOF){
iterations=1000000; }
            }
      }

      // Determine multiplication factor of k in order to use fixed
point notation
      if(kl < 1){
            MF = 10000000;
            if(kl < 0.0000001){
                  MF = (unsigned int)(1.0/kl);
            }
            kl_int = (unsigned int)(kl * MF);
            if((unsigned int)(kl * MF * 10)%10 >=5) kl_int += 1;
            for(i=0;i<6;i++){
                  if(kl_int %10 > 0) break;
                  MF /= 10;
                  kl_int /= 10;
            }
      }

      //Update fixed point k values for each reaction
      for(i=0;i<m;i++){
```

```
            R[i].fpk = (unsigned int)(R[i].k * MF);
            if((unsigned int)(R[i].k*MF * 10)%10 >= 5) R[i].fpk +=1;
            if(R[i].fpk>KMAX){
                  fprintf(stderr,"ERROR!  Rate constant of reaction %d
exceeds maximum of %d\n",i,KMAX);
                  exit(1);
            }
      }

      setseed(seed);
      thecount = 0;
      thetime = 0.0;

      /*
      fprintf(outFile,"%6d  %8.6lf",thecount,thetime);
      for(j=0;j<num;j++){
            fprintf(outFile,"  %6u",X[mon[i]]);
      }
      fprintf(outFile,"\n");
      thecount++;
      */

      // Send initial species populations
      for(i=0;i<n;i++){
            setspeciespop(i,X[i]);
      }
      for(i;i<NMAX;i++){
            setspeciespop(i,0);
      }

      // Send reaction equations
      for(i=0;i<m;i++){
            for(j=0;j<4;j++){ reacdata[j]=NULLSPECIES; }
            j=0;
            for(k=0;k<R[i].reactants;k++){
                  for(l=0;l<R[i].rewt[k];l++){
                        reacdata[j++]=R[i].renum[k];
                  }
            }
            if(reacdata[1]==NULLSPECIES){
                  reacdata[1]=reacdata[0];
                  reacdata[0]=NULLSPECIES;
            }
            j=2;
            for(k=0;k<R[i].products;k++){
                  for(l=0;l<R[i].prwt[k];l++){
                        reacdata[j++]=R[i].prnum[k];
                  }
            }

      setreaction(i,reacdata[0],reacdata[1],reacdata[2],reacdata[3],R[i
].fpk);
      }
      for(i;i<MMAX;i++){

      setreaction(i,NULLSPECIES,NULLSPECIES,NULLSPECIES,NULLSPECIES,0);
      }
```

```
	step(iterations);

	gettimeofday(&te,NULL);
	printf("Run Time: %f\n",(double)(te.tv_sec-
ts.tv_sec)+0.000001*(double)(te.tv_usec-ts.tv_usec));

	for(i=0;i<n;i++){
		printf("Species %d: %d\n",i,readspeciespop(i));
	}

	munmap(memp, MTRRZ);
	close(fd);

	return 0;
}
```

Appendix C

BRAM Based Design VHDL

parith.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY parith IS
  PORT (    clk   : IN STD_LOGIC;
            we    : OUT STD_LOGIC;
            addr  : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            din   : OUT STD_LOGIC_VECTOR(63 DOWNTO 0);
            dout  : IN STD_LOGIC_VECTOR(63 DOWNTO 0));
END parith;

ARCHITECTURE rtl OF parith IS

COMPONENT lfsr32
      PORT (
            in_clock    : IN STD_LOGIC;
            in_reset    : IN STD_LOGIC;
            in_seed           : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
            out_random_number : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END COMPONENT;

COMPONENT exp_rand
      PORT (
            in_clock    : IN STD_LOGIC;
            out_uniform_number      : OUT STD_LOGIC_VECTOR(31 DOWNTO
0);
            out_random_number : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END COMPONENT;

COMPONENT sumprop
      PORT (
            clk         : IN STD_LOGIC;
            PSUM1       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM2       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM3       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM4       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM5       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM6       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM7       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM8       : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL2            : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL3            : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL4            : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL5            : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL6            : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL7            : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL8            : OUT STD_LOGIC_VECTOR(47 DOWNTO 0) );
END COMPONENT;
```

```
COMPONENT propcalc
      PORT (
            clk         : IN STD_LOGIC;
            POP1        : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            POP2        : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            RX          : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PROPENSITY  : OUT STD_LOGIC_VECTOR(47 DOWNTO 0) );
END COMPONENT;

COMPONENT dpram16_128
      PORT (
            addra : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
            addrb : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
            clka  : IN STD_LOGIC;
            clkb  : IN STD_LOGIC;
            dina  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            dinb  : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            douta : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
            doutb : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
            wea         : IN STD_LOGIC;
            web         : IN STD_LOGIC );
END COMPONENT;

COMPONENT dpram48_64
      PORT (
            addra : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
            addrb : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
            clka  : IN STD_LOGIC;
            clkb  : IN STD_LOGIC;
            dina  : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            dinb  : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            douta : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            doutb : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            wea         : IN STD_LOGIC;
            web         : IN STD_LOGIC );
END COMPONENT;

SIGNAL s_lfsr_enable    : STD_LOGIC;
SIGNAL s_lfsr_reset           : STD_LOGIC;
SIGNAL s_seed                 : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL s_URV                  : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL s_rxselect       : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL s_ERV_URV        : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL s_ERV                  : STD_LOGIC_VECTOR(31 DOWNTO 0);

SIGNAL SP1a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP1a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP1a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP1a_wea         : STD_LOGIC;
SIGNAL SP1a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP1a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP1a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP1a_web         : STD_LOGIC;

SIGNAL SP1b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP1b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
```

```
SIGNAL SP1b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP1b_wea         : STD_LOGIC;
SIGNAL SP1b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP1b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP1b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP1b_web         : STD_LOGIC;

SIGNAL SP2a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP2a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2a_wea         : STD_LOGIC;
SIGNAL SP2a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP2a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2a_web         : STD_LOGIC;

SIGNAL SP2b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP2b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2b_wea         : STD_LOGIC;
SIGNAL SP2b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP2b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP2b_web         : STD_LOGIC;

SIGNAL SP3a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP3a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3a_wea         : STD_LOGIC;
SIGNAL SP3a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP3a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3a_web         : STD_LOGIC;

SIGNAL SP3b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP3b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3b_wea         : STD_LOGIC;
SIGNAL SP3b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP3b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP3b_web         : STD_LOGIC;

SIGNAL SP4a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP4a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4a_wea         : STD_LOGIC;
SIGNAL SP4a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP4a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4a_web         : STD_LOGIC;

SIGNAL SP4b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP4b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4b_wea         : STD_LOGIC;
SIGNAL SP4b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
```

```
SIGNAL SP4b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP4b_web         : STD_LOGIC;

SIGNAL SP5a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP5a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5a_wea         : STD_LOGIC;
SIGNAL SP5a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP5a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5a_web         : STD_LOGIC;

SIGNAL SP5b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP5b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5b_wea         : STD_LOGIC;
SIGNAL SP5b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP5b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP5b_web         : STD_LOGIC;

SIGNAL SP6a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP6a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6a_wea         : STD_LOGIC;
SIGNAL SP6a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP6a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6a_web         : STD_LOGIC;

SIGNAL SP6b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP6b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6b_wea         : STD_LOGIC;
SIGNAL SP6b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP6b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP6b_web         : STD_LOGIC;

SIGNAL SP7a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP7a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7a_wea         : STD_LOGIC;
SIGNAL SP7a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP7a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7a_web         : STD_LOGIC;

SIGNAL SP7b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP7b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7b_wea         : STD_LOGIC;
SIGNAL SP7b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP7b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP7b_web         : STD_LOGIC;
```

```
SIGNAL SP8a_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP8a_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8a_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8a_wea         : STD_LOGIC;
SIGNAL SP8a_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP8a_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8a_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8a_web         : STD_LOGIC;

SIGNAL SP8b_addra : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP8b_dina  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8b_douta : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8b_wea         : STD_LOGIC;
SIGNAL SP8b_addrb : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SP8b_dinb  : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8b_doutb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SP8b_web         : STD_LOGIC;

SIGNAL SPUS1_addra      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS1_dina : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS1_douta      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS1_wea        : STD_LOGIC;
SIGNAL SPUS1_addrb      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS1_dinb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS1_doutb      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS1_web        : STD_LOGIC;
SIGNAL SPUS2_addra      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS2_dina : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS2_douta      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS2_wea        : STD_LOGIC;
SIGNAL SPUS2_addrb      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS2_dinb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS2_doutb      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS2_web        : STD_LOGIC;
SIGNAL SPUS3_addra      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS3_dina : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS3_douta      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS3_wea        : STD_LOGIC;
SIGNAL SPUS3_addrb      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS3_dinb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS3_doutb      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS3_web        : STD_LOGIC;
SIGNAL SPUS4_addra      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS4_dina : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS4_douta      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS4_wea        : STD_LOGIC;
SIGNAL SPUS4_addrb      : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL SPUS4_dinb : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS4_doutb      : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL SPUS4_web        : STD_LOGIC;

SIGNAL RX1_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX1_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX1_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX1_wea    : STD_LOGIC;
SIGNAL RX1_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
```

```
SIGNAL RX1_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX1_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX1_web    : STD_LOGIC;
SIGNAL RX2_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX2_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX2_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX2_wea    : STD_LOGIC;
SIGNAL RX2_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX2_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX2_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX2_web    : STD_LOGIC;
SIGNAL RX3_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX3_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX3_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX3_wea    : STD_LOGIC;
SIGNAL RX3_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX3_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX3_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX3_web    : STD_LOGIC;
SIGNAL RX4_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX4_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX4_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX4_wea    : STD_LOGIC;
SIGNAL RX4_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX4_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX4_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX4_web    : STD_LOGIC;
SIGNAL RX5_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX5_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX5_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX5_wea    : STD_LOGIC;
SIGNAL RX5_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX5_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX5_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX5_web    : STD_LOGIC;
SIGNAL RX6_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX6_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX6_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX6_wea    : STD_LOGIC;
SIGNAL RX6_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX6_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX6_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX6_web    : STD_LOGIC;
SIGNAL RX7_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX7_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX7_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX7_wea    : STD_LOGIC;
SIGNAL RX7_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX7_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX7_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX7_web    : STD_LOGIC;
SIGNAL RX8_addra  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX8_dina   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX8_douta  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX8_wea    : STD_LOGIC;
SIGNAL RX8_addrb  : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RX8_dinb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
```

```
SIGNAL RX8_doutb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RX8_web    : STD_LOGIC;
SIGNAL RXUS_addra : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RXUS_dina  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RXUS_douta : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RXUS_wea   : STD_LOGIC;
SIGNAL RXUS_addrb : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL RXUS_dinb  : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RXUS_doutb : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL RXUS_web   : STD_LOGIC;

SIGNAL P1_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P1_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P1_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P1_wea     : STD_LOGIC;
SIGNAL P1_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P1_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P1_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P1_web     : STD_LOGIC;
SIGNAL P2_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P2_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P2_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P2_wea     : STD_LOGIC;
SIGNAL P2_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P2_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P2_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P2_web     : STD_LOGIC;
SIGNAL P3_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P3_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P3_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P3_wea     : STD_LOGIC;
SIGNAL P3_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P3_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P3_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P3_web     : STD_LOGIC;
SIGNAL P4_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P4_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P4_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P4_wea     : STD_LOGIC;
SIGNAL P4_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P4_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P4_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P4_web     : STD_LOGIC;
SIGNAL P5_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P5_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P5_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P5_wea     : STD_LOGIC;
SIGNAL P5_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P5_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P5_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P5_web     : STD_LOGIC;
SIGNAL P6_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P6_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P6_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P6_wea     : STD_LOGIC;
SIGNAL P6_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P6_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
```

```
SIGNAL P6_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P6_web     : STD_LOGIC;
SIGNAL P7_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P7_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P7_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P7_wea     : STD_LOGIC;
SIGNAL P7_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P7_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P7_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P7_web     : STD_LOGIC;
SIGNAL P8_addra   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P8_dina    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P8_douta   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P8_wea     : STD_LOGIC;
SIGNAL P8_addrb   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL P8_dinb    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P8_doutb   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL P8_web     : STD_LOGIC;

SIGNAL PSUM1_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM1_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM2_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM2_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM3_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM3_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM4_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM4_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM5_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM5_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM6_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM6_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM7_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM7_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM8_1    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PSUM8_2    : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL TPROP2     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL TPROP3     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL TPROP4     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL TPROP5     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL TPROP6     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL TPROP7     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL TPROP8     : STD_LOGIC_VECTOR(47 DOWNTO 0);

SIGNAL LBOUND_1   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL LBOUND_2   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL LBOUND_3   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL LBOUND_4   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL LBOUND_5   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL LBOUND_6   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL LBOUND_7   : STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL LBOUND_8   : STD_LOGIC_VECTOR(5 DOWNTO 0);

SIGNAL PC1_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC1_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC1_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC1_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC2_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
```

```
SIGNAL PC2_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC2_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC2_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC3_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC3_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC3_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC3_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC4_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC4_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC4_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC4_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC5_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC5_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC5_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC5_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC6_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC6_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC6_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC6_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC7_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC7_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC7_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC7_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC8_POP1   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC8_POP2   : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL PC8_RX     : STD_LOGIC_VECTOR(47 DOWNTO 0);
SIGNAL PC8_PROP   : STD_LOGIC_VECTOR(47 DOWNTO 0);

SIGNAL R1I        : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL R1V        : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL R2I        : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL R2V        : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL P1I        : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL P1V        : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL P2I        : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL P2V        : STD_LOGIC_VECTOR(15 DOWNTO 0);

SIGNAL product    : STD_LOGIC_VECTOR(79 DOWNTO 0);

BEGIN

  ERV : exp_rand  PORT MAP (
      in_clock => clk,
      out_uniform_number => s_ERV_URV,
      out_random_number => s_ERV
  );

  URV : lfsr32 PORT MAP (
      in_clock => s_lfsr_enable,
      in_reset => s_lfsr_reset,
      in_seed => s_seed,
      out_random_number => s_URV
  );

  SP1adpram : dpram16_128 PORT MAP (
      addra => SP1a_addra,
      addrb => SP1a_addrb,
```

```
    clka => clk,
    clkb => clk,
    dina => SP1a_dina,
    dinb => SP1a_dinb,
    douta => SP1a_douta,
    doutb => SP1a_doutb,
    wea => SP1a_wea,
    web => SP1a_web
);

SP1bdpram : dpram16_128 PORT MAP (
    addra => SP1b_addra,
    addrb => SP1b_addrb,
    clka => clk,
    clkb => clk,
    dina => SP1b_dina,
    dinb => SP1b_dinb,
    douta => SP1b_douta,
    doutb => SP1b_doutb,
    wea => SP1b_wea,
    web => SP1b_web
);

SP2adpram : dpram16_128 PORT MAP (
    addra => SP2a_addra,
    addrb => SP2a_addrb,
    clka => clk,
    clkb => clk,
    dina => SP2a_dina,
    dinb => SP2a_dinb,
    douta => SP2a_douta,
    doutb => SP2a_doutb,
    wea => SP2a_wea,
    web => SP2a_web
);

SP2bdpram : dpram16_128 PORT MAP (
    addra => SP2b_addra,
    addrb => SP2b_addrb,
    clka => clk,
    clkb => clk,
    dina => SP2b_dina,
    dinb => SP2b_dinb,
    douta => SP2b_douta,
    doutb => SP2b_doutb,
    wea => SP2b_wea,
    web => SP2b_web
);

SP3adpram : dpram16_128 PORT MAP (
    addra => SP3a_addra,
    addrb => SP3a_addrb,
    clka => clk,
    clkb => clk,
    dina => SP3a_dina,
    dinb => SP3a_dinb,
    douta => SP3a_douta,
```

```
    doutb => SP3a_doutb,
    wea => SP3a_wea,
    web => SP3a_web
);

SP3bdpram : dpram16_128 PORT MAP (
    addra => SP3b_addra,
    addrb => SP3b_addrb,
    clka => clk,
    clkb => clk,
    dina => SP3b_dina,
    dinb => SP3b_dinb,
    douta => SP3b_douta,
    doutb => SP3b_doutb,
    wea => SP3b_wea,
    web => SP3b_web
);

SP4adpram : dpram16_128 PORT MAP (
    addra => SP4a_addra,
    addrb => SP4a_addrb,
    clka => clk,
    clkb => clk,
    dina => SP4a_dina,
    dinb => SP4a_dinb,
    douta => SP4a_douta,
    doutb => SP4a_doutb,
    wea => SP4a_wea,
    web => SP4a_web
);

SP4bdpram : dpram16_128 PORT MAP (
    addra => SP4b_addra,
    addrb => SP4b_addrb,
    clka => clk,
    clkb => clk,
    dina => SP4b_dina,
    dinb => SP4b_dinb,
    douta => SP4b_douta,
    doutb => SP4b_doutb,
    wea => SP4b_wea,
    web => SP4b_web
);

SP5adpram : dpram16_128 PORT MAP (
    addra => SP5a_addra,
    addrb => SP5a_addrb,
    clka => clk,
    clkb => clk,
    dina => SP5a_dina,
    dinb => SP5a_dinb,
    douta => SP5a_douta,
    doutb => SP5a_doutb,
    wea => SP5a_wea,
    web => SP5a_web
);
```

```
SP5bdpram : dpram16_128 PORT MAP (
    addra => SP5b_addra,
    addrb => SP5b_addrb,
    clka => clk,
    clkb => clk,
    dina => SP5b_dina,
    dinb => SP5b_dinb,
    douta => SP5b_douta,
    doutb => SP5b_doutb,
    wea => SP5b_wea,
    web => SP5b_web
);

SP6adpram : dpram16_128 PORT MAP (
    addra => SP6a_addra,
    addrb => SP6a_addrb,
    clka => clk,
    clkb => clk,
    dina => SP6a_dina,
    dinb => SP6a_dinb,
    douta => SP6a_douta,
    doutb => SP6a_doutb,
    wea => SP6a_wea,
    web => SP6a_web
);

SP6bdpram : dpram16_128 PORT MAP (
    addra => SP6b_addra,
    addrb => SP6b_addrb,
    clka => clk,
    clkb => clk,
    dina => SP6b_dina,
    dinb => SP6b_dinb,
    douta => SP6b_douta,
    doutb => SP6b_doutb,
    wea => SP6b_wea,
    web => SP6b_web
);

SP7adpram : dpram16_128 PORT MAP (
    addra => SP7a_addra,
    addrb => SP7a_addrb,
    clka => clk,
    clkb => clk,
    dina => SP7a_dina,
    dinb => SP7a_dinb,
    douta => SP7a_douta,
    doutb => SP7a_doutb,
    wea => SP7a_wea,
    web => SP7a_web
);

SP7bdpram : dpram16_128 PORT MAP (
    addra => SP7b_addra,
    addrb => SP7b_addrb,
    clka => clk,
    clkb => clk,
```

```
    dina => SP7b_dina,
    dinb => SP7b_dinb,
    douta => SP7b_douta,
    doutb => SP7b_doutb,
    wea => SP7b_wea,
    web => SP7b_web
);

SP8adpram : dpram16_128 PORT MAP (
    addra => SP8a_addra,
    addrb => SP8a_addrb,
    clka => clk,
    clkb => clk,
    dina => SP8a_dina,
    dinb => SP8a_dinb,
    douta => SP8a_douta,
    doutb => SP8a_doutb,
    wea => SP8a_wea,
    web => SP8a_web
);

SP8bdpram : dpram16_128 PORT MAP (
    addra => SP8b_addra,
    addrb => SP8b_addrb,
    clka => clk,
    clkb => clk,
    dina => SP8b_dina,
    dinb => SP8b_dinb,
    douta => SP8b_douta,
    doutb => SP8b_doutb,
    wea => SP8b_wea,
    web => SP8b_web
);

SPUS1dpram : dpram16_128 PORT MAP (
    addra => SPUS1_addra,
    addrb => SPUS1_addrb,
    clka => clk,
    clkb => clk,
    dina => SPUS1_dina,
    dinb => SPUS1_dinb,
    douta => SPUS1_douta,
    doutb => SPUS1_doutb,
    wea => SPUS1_wea,
    web => SPUS1_web
);

SPUS2dpram : dpram16_128 PORT MAP (
    addra => SPUS2_addra,
    addrb => SPUS2_addrb,
    clka => clk,
    clkb => clk,
    dina => SPUS2_dina,
    dinb => SPUS2_dinb,
    douta => SPUS2_douta,
    doutb => SPUS2_doutb,
    wea => SPUS2_wea,
```

```
    web => SPUS2_web
);

SPUS3dpram : dpram16_128 PORT MAP (
    addra => SPUS3_addra,
    addrb => SPUS3_addrb,
    clka => clk,
    clkb => clk,
    dina => SPUS3_dina,
    dinb => SPUS3_dinb,
    douta => SPUS3_douta,
    doutb => SPUS3_doutb,
    wea => SPUS3_wea,
    web => SPUS3_web
);

SPUS4dpram : dpram16_128 PORT MAP (
    addra => SPUS4_addra,
    addrb => SPUS4_addrb,
    clka => clk,
    clkb => clk,
    dina => SPUS4_dina,
    dinb => SPUS4_dinb,
    douta => SPUS4_douta,
    doutb => SPUS4_doutb,
    wea => SPUS4_wea,
    web => SPUS4_web
);

RX1dpram : dpram48_64 PORT MAP (
    addra => RX1_addra,
    addrb => RX1_addrb,
    clka => clk,
    clkb => clk,
    dina => RX1_dina,
    dinb => RX1_dinb,
    douta => RX1_douta,
    doutb => RX1_doutb,
    wea => RX1_wea,
    web => RX1_web
);
RX2dpram : dpram48_64 PORT MAP (
    addra => RX2_addra,
    addrb => RX2_addrb,
    clka => clk,
    clkb => clk,
    dina => RX2_dina,
    dinb => RX2_dinb,
    douta => RX2_douta,
    doutb => RX2_doutb,
    wea => RX2_wea,
    web => RX2_web
);

RX3dpram : dpram48_64 PORT MAP (
    addra => RX3_addra,
    addrb => RX3_addrb,
```

```
    clka => clk,
    clkb => clk,
    dina => RX3_dina,
    dinb => RX3_dinb,
    douta => RX3_douta,
    doutb => RX3_doutb,
    wea => RX3_wea,
    web => RX3_web
);

RX4dpram : dpram48_64 PORT MAP (
    addra => RX4_addra,
    addrb => RX4_addrb,
    clka => clk,
    clkb => clk,
    dina => RX4_dina,
    dinb => RX4_dinb,
    douta => RX4_douta,
    doutb => RX4_doutb,
    wea => RX4_wea,
    web => RX4_web
);

RX5dpram : dpram48_64 PORT MAP (
    addra => RX5_addra,
    addrb => RX5_addrb,
    clka => clk,
    clkb => clk,
    dina => RX5_dina,
    dinb => RX5_dinb,
    douta => RX5_douta,
    doutb => RX5_doutb,
    wea => RX5_wea,
    web => RX5_web
);

RX6dpram : dpram48_64 PORT MAP (
    addra => RX6_addra,
    addrb => RX6_addrb,
    clka => clk,
    clkb => clk,
    dina => RX6_dina,
    dinb => RX6_dinb,
    douta => RX6_douta,
    doutb => RX6_doutb,
    wea => RX6_wea,
    web => RX6_web
);

RX7dpram : dpram48_64 PORT MAP (
    addra => RX7_addra,
    addrb => RX7_addrb,
    clka => clk,
    clkb => clk,
    dina => RX7_dina,
    dinb => RX7_dinb,
    douta => RX7_douta,
```

```
    doutb => RX7_doutb,
    wea => RX7_wea,
    web => RX7_web
);

RX8dpram : dpram48_64 PORT MAP (
    addra => RX8_addra,
    addrb => RX8_addrb,
    clka => clk,
    clkb => clk,
    dina => RX8_dina,
    dinb => RX8_dinb,
    douta => RX8_douta,
    doutb => RX8_doutb,
    wea => RX8_wea,
    web => RX8_web
);

RXUSdpram : dpram48_64 PORT MAP (
    addra => RXUS_addra,
    addrb => RXUS_addrb,
    clka => clk,
    clkb => clk,
    dina => RXUS_dina,
    dinb => RXUS_dinb,
    douta => RXUS_douta,
    doutb => RXUS_doutb,
    wea => RXUS_wea,
    web => RXUS_web
);

P1dpram : dpram48_64 PORT MAP (
    addra => P1_addra,
    addrb => P1_addrb,
    clka => clk,
    clkb => clk,
    dina => P1_dina,
    dinb => P1_dinb,
    douta => P1_douta,
    doutb => P1_doutb,
    wea => P1_wea,
    web => P1_web
);

P2dpram : dpram48_64 PORT MAP (
    addra => P2_addra,
    addrb => P2_addrb,
    clka => clk,
    clkb => clk,
    dina => P2_dina,
    dinb => P2_dinb,
    douta => P2_douta,
    doutb => P2_doutb,
    wea => P2_wea,
    web => P2_web
);
```

```
P3dpram : dpram48_64 PORT MAP (
    addra => P3_addra,
    addrb => P3_addrb,
    clka => clk,
    clkb => clk,
    dina => P3_dina,
    dinb => P3_dinb,
    douta => P3_douta,
    doutb => P3_doutb,
    wea => P3_wea,
    web => P3_web
);

P4dpram : dpram48_64 PORT MAP (
    addra => P4_addra,
    addrb => P4_addrb,
    clka => clk,
    clkb => clk,
    dina => P4_dina,
    dinb => P4_dinb,
    douta => P4_douta,
    doutb => P4_doutb,
    wea => P4_wea,
    web => P4_web
);

P5dpram : dpram48_64 PORT MAP (
    addra => P5_addra,
    addrb => P5_addrb,
    clka => clk,
    clkb => clk,
    dina => P5_dina,
    dinb => P5_dinb,
    douta => P5_douta,
    doutb => P5_doutb,
    wea => P5_wea,
    web => P5_web
);

P6dpram : dpram48_64 PORT MAP (
    addra => P6_addra,
    addrb => P6_addrb,
    clka => clk,
    clkb => clk,
    dina => P6_dina,
    dinb => P6_dinb,
    douta => P6_douta,
    doutb => P6_doutb,
    wea => P6_wea,
    web => P6_web
);

P7dpram : dpram48_64 PORT MAP (
    addra => P7_addra,
    addrb => P7_addrb,
    clka => clk,
    clkb => clk,
```

```
    dina => P7_dina,
    dinb => P7_dinb,
    douta => P7_douta,
    doutb => P7_doutb,
    wea => P7_wea,
    web => P7_web
);

P8dpram : dpram48_64 PORT MAP (
    addra => P8_addra,
    addrb => P8_addrb,
    clka => clk,
    clkb => clk,
    dina => P8_dina,
    dinb => P8_dinb,
    douta => P8_douta,
    doutb => P8_doutb,
    wea => P8_wea,
    web => P8_web
);

PROP1 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC1_POP1,
    POP2 => PC1_POP2,
    RX => PC1_RX,
    PROPENSITY => PC1_PROP
);

PROP2 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC2_POP1,
    POP2 => PC2_POP2,
    RX => PC2_RX,
    PROPENSITY => PC2_PROP
);

PROP3 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC3_POP1,
    POP2 => PC3_POP2,
    RX => PC3_RX,
    PROPENSITY => PC3_PROP
);

PROP4 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC4_POP1,
    POP2 => PC4_POP2,
    RX => PC4_RX,
    PROPENSITY => PC4_PROP
);

PROP5 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC5_POP1,
    POP2 => PC5_POP2,
```

```
    RX => PC5_RX,
    PROPENSITY => PC5_PROP
);

PROP6 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC6_POP1,
    POP2 => PC6_POP2,
    RX => PC6_RX,
    PROPENSITY => PC6_PROP
);

PROP7 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC7_POP1,
    POP2 => PC7_POP2,
    RX => PC7_RX,
    PROPENSITY => PC7_PROP
);

PROP8 : propcalc PORT MAP (
    clk => clk,
    POP1 => PC8_POP1,
    POP2 => PC8_POP2,
    RX => PC8_RX,
    PROPENSITY => PC8_PROP
);

TOTALPROP : sumprop PORT MAP (
    clk => clk,
    PSUM1 => PSUM1_2,
    PSUM2 => PSUM2_2,
    PSUM3 => PSUM3_2,
    PSUM4 => PSUM4_2,
    PSUM5 => PSUM5_2,
    PSUM6 => PSUM6_2,
    PSUM7 => PSUM7_2,
    PSUM8 => PSUM8_2,
    TOTAL2 => TPROP2,
    TOTAL3 => TPROP3,
    TOTAL4 => TPROP4,
    TOTAL5 => TPROP5,
    TOTAL6 => TPROP6,
    TOTAL7 => TPROP7,
    TOTAL8 => TPROP8
);

PROCESS (clk)
    VARIABLE state    : STD_LOGIC_VECTOR(7 DOWNTO 0);
    VARIABLE state2   : STD_LOGIC_VECTOR(7 DOWNTO 0);
    VARIABLE count    : STD_LOGIC_VECTOR(5 DOWNTO 0);
    VARIABLE looping  : STD_LOGIC;
    VARIABLE index    : STD_LOGIC_VECTOR(7 DOWNTO 0);
    VARIABLE maxindex : STD_LOGIC_VECTOR(7 DOWNTO 0);
    VARIABLE theproduct     : STD_LOGIC_VECTOR(79 DOWNTO 0);

    VARIABLE v_PSUM1_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
```

```
    VARIABLE v_PSUM1_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM2_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM2_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM3_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM3_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM4_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM4_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM5_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM5_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM6_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM6_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM7_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM7_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM8_1      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PSUM8_2      : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC1_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC1_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC1_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC2_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC2_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC2_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC3_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC3_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC3_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC4_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC4_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC4_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC5_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC5_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC5_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC6_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC6_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC6_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC7_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC7_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC7_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_PC8_POP1     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC8_POP2     : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_PC8_RX       : STD_LOGIC_VECTOR(47 DOWNTO 0);
    VARIABLE v_RX                 : STD_LOGIC_VECTOR(5 DOWNTO 0);
    VARIABLE v_R1V                : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_R2V                : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_P1V                : STD_LOGIC_VECTOR(15 DOWNTO 0);
    VARIABLE v_P2V                : STD_LOGIC_VECTOR(15 DOWNTO 0);

BEGIN
    IF (clk = '0' AND clk'EVENT) THEN
          LBOUND_1 <= "000000";
          LBOUND_2 <= "001000";
          LBOUND_3 <= "010000";
          LBOUND_4 <= "011000";
          LBOUND_5 <= "100000";
          LBOUND_6 <= "101000";
          LBOUND_7 <= "110000";
          LBOUND_8 <= "111000";

          theproduct := s_URV * TPROP8;
```

```
-- SET ADDRESS FROM WHICH TO READ COMMAND
IF (state = "00000000") THEN
      we <= '0';
      addr <= X"00";
      din <= (others => '0');

      s_lfsr_reset <= '0';
      s_lfsr_enable <= '0';

      index := X"02";
      maxindex := X"FC";
      looping := '0';
      count := "000000";

      SPUS1_wea <= '0';
      SPUS1_addra <= "0000000";
      SPUS1_dina <= (others => '0');
      SPUS2_wea <= '0';
      SPUS2_addra <= "0000000";
      SPUS2_dina <= (others => '0');
      SPUS3_wea <= '0';
      SPUS3_addra <= "0000000";
      SPUS3_dina <= (others => '0');
      SPUS4_wea <= '0';
      SPUS4_addra <= "0000000";
      SPUS4_dina <= (others => '0');
      SP1a_wea <= '0';
      SP1a_addra <= "0000000";
      SP1a_dina <= (others => '0');
      SP1b_wea <= '0';
      SP1b_addra <= "0000000";
      SP1b_dina <= (others => '0');
      RX1_wea <= '0';
      RX1_addra <= "000000";
      RX1_dina <= (others => '0');
      P1_wea <= '0';
      P1_addra <= "000000";
      P1_dina <= (others => '0');
      SP2a_wea <= '0';
      SP2a_addra <= "0000000";
      SP2a_dina <= (others => '0');
      SP2b_wea <= '0';
      SP2b_addra <= "0000000";
      SP2b_dina <= (others => '0');
      RX2_wea <= '0';
      RX2_addra <= "000000";
      RX2_dina <= (others => '0');
      P2_wea <= '0';
      P2_addra <= "000000";
      P2_dina <= (others => '0');
      SP3a_wea <= '0';
      SP3a_addra <= "0000000";
      SP3a_dina <= (others => '0');
      SP3b_wea <= '0';
      SP3b_addra <= "0000000";
      SP3b_dina <= (others => '0');
```

```
RX3_wea <= '0';
RX3_addra <= "000000";
RX3_dina <= (others => '0');
P3_wea <= '0';
P3_addra <= "000000";
P3_dina <= (others => '0');
SP4a_wea <= '0';
SP4a_addra <= "0000000";
SP4a_dina <= (others => '0');
SP4b_wea <= '0';
SP4b_addra <= "0000000";
SP4b_dina <= (others => '0');
RX4_wea <= '0';
RX4_addra <= "000000";
RX4_dina <= (others => '0');
P4_wea <= '0';
P4_addra <= "000000";
P4_dina <= (others => '0');
SP5a_wea <= '0';
SP5a_addra <= "0000000";
SP5a_dina <= (others => '0');
SP5b_wea <= '0';
SP5b_addra <= "0000000";
SP5b_dina <= (others => '0');
RX5_wea <= '0';
RX5_addra <= "000000";
RX5_dina <= (others => '0');
P5_wea <= '0';
P5_addra <= "000000";
P5_dina <= (others => '0');
SP6a_wea <= '0';
SP6a_addra <= "0000000";
SP6a_dina <= (others => '0');
SP6b_wea <= '0';
SP6b_addra <= "0000000";
SP6b_dina <= (others => '0');
RX6_wea <= '0';
RX6_addra <= "000000";
RX6_dina <= (others => '0');
P6_wea <= '0';
P6_addra <= "000000";
P6_dina <= (others => '0');
SP7a_wea <= '0';
SP7a_addra <= "0000000";
SP7a_dina <= (others => '0');
SP7b_wea <= '0';
SP7b_addra <= "0000000";
SP7b_dina <= (others => '0');
RX7_wea <= '0';
RX7_addra <= "000000";
RX7_dina <= (others => '0');
P7_wea <= '0';
P7_addra <= "000000";
P7_dina <= (others => '0');
SP8a_wea <= '0';
SP8a_addra <= "0000000";
SP8a_dina <= (others => '0');
```

```
        SP8b_wea <= '0';
        SP8b_addra <= "0000000";
        SP8b_dina <= (others => '0');
        RX8_wea <= '0';
        RX8_addra <= "000000";
        RX8_dina <= (others => '0');
        P8_wea <= '0';
        P8_addra <= "000000";
        P8_dina <= (others => '0');
        RXUS_wea <= '0';
        RXUS_addra <= "000000";
        RXUS_dina <= (others => '0');

        state := state + 1;
        state2 := "00000000";

    -- INTERPRET COMMANDS
    ELSIF (state = "00000001") THEN

        -- LOOPING THROUGH 250 REACTIONS
        IF (looping = '1') THEN
            IF (index < maxindex) THEN
                IF (state2 = "00000000") THEN
                    we <= '0';
                    addr <= X"00";
                    P1_wea <= '0';
                    P2_wea <= '0';
                    P3_wea <= '0';
                    P4_wea <= '0';
                    P5_wea <= '0';
                    P6_wea <= '0';
                    P7_wea <= '0';
                    P8_wea <= '0';

                    SPUS1_wea <= '0';
                    SPUS2_wea <= '0';
                    SPUS3_wea <= '0';
                    SPUS4_wea <= '0';
                    SP1a_wea <= '0';
                    SP1b_wea <= '0';
                    SP2a_wea <= '0';
                    SP2b_wea <= '0';
                    SP3a_wea <= '0';
                    SP3b_wea <= '0';
                    SP4a_wea <= '0';
                    SP4b_wea <= '0';
                    SP5a_wea <= '0';
                    SP5b_wea <= '0';
                    SP6a_wea <= '0';
                    SP6b_wea <= '0';
                    SP7a_wea <= '0';
                    SP7b_wea <= '0';
                    SP8a_wea <= '0';
                    SP8b_wea <= '0';

                    RX1_wea <= '0';
                    RX1_addra <= LBOUND_1 + count;
```

```
                RX2_wea <= '0';
                RX2_addra <= LBOUND_2 + count;
                RX3_wea <= '0';
                RX3_addra <= LBOUND_3 + count;
                RX4_wea <= '0';
                RX4_addra <= LBOUND_4 + count;
                RX5_wea <= '0';
                RX5_addra <= LBOUND_5 + count;
                RX6_wea <= '0';
                RX6_addra <= LBOUND_6 + count;
                RX7_wea <= '0';
                RX7_addra <= LBOUND_7 + count;
                RX8_wea <= '0';
                RX8_addra <= LBOUND_8 + count;
                state2 := state2 + 1;

            ELSIF (state2 = "00000001") THEN
                we <= '0';
                addr <= X"00";
                RX1_wea <= '0';
                RX1_addra <= LBOUND_1 + count;
                RX2_wea <= '0';
                RX2_addra <= LBOUND_2 + count;
                RX3_wea <= '0';
                RX3_addra <= LBOUND_3 + count;
                RX4_wea <= '0';
                RX4_addra <= LBOUND_4 + count;
                RX5_wea <= '0';
                RX5_addra <= LBOUND_5 + count;
                RX6_wea <= '0';
                RX6_addra <= LBOUND_6 + count;
                RX7_wea <= '0';
                RX7_addra <= LBOUND_7 + count;
                RX8_wea <= '0';
                RX8_addra <= LBOUND_8 + count;

                SP1a_wea <= '0';
                SP1b_wea <= '0';
                SP1a_addra <= RX1_douta(46 DOWNTO
40);
                SP1b_addra <= RX1_douta(38 DOWNTO
32);
                SP2a_wea <= '0';
                SP2b_wea <= '0';
                SP2a_addra <= RX2_douta(46 DOWNTO
40);
                SP2b_addra <= RX2_douta(38 DOWNTO
32);
                SP3a_wea <= '0';
                SP3b_wea <= '0';
                SP3a_addra <= RX3_douta(46 DOWNTO
40);
                SP3b_addra <= RX3_douta(38 DOWNTO
32);
                SP4a_wea <= '0';
                SP4b_wea <= '0';
```

```
                        SP4a_addra <= RX4_douta(46 DOWNTO
40);
                        SP4b_addra <= RX4_douta(38 DOWNTO
32);
                        SP5a_wea <= '0';
                        SP5b_wea <= '0';
                        SP5a_addra <= RX5_douta(46 DOWNTO
40);
                        SP5b_addra <= RX5_douta(38 DOWNTO
32);
                        SP6a_wea <= '0';
                        SP6b_wea <= '0';
                        SP6a_addra <= RX6_douta(46 DOWNTO
40);
                        SP6b_addra <= RX6_douta(38 DOWNTO
32);
                        SP7a_wea <= '0';
                        SP7b_wea <= '0';
                        SP7a_addra <= RX7_douta(46 DOWNTO
40);
                        SP7b_addra <= RX7_douta(38 DOWNTO
32);
                        SP8a_wea <= '0';
                        SP8b_wea <= '0';
                        SP8a_addra <= RX8_douta(46 DOWNTO
40);
                        SP8b_addra <= RX8_douta(38 DOWNTO
32);
                        state2 := state2 + 1;

                    ELSIF (state2 = "00000010") THEN
                        we <= '0';
                        addr <= X"00";
                        RX1_wea <= '0';
                        RX1_addra <= LBOUND_1 + count;
                        RX2_wea <= '0';
                        RX2_addra <= LBOUND_2 + count;
                        RX3_wea <= '0';
                        RX3_addra <= LBOUND_3 + count;
                        RX4_wea <= '0';
                        RX4_addra <= LBOUND_4 + count;
                        RX5_wea <= '0';
                        RX5_addra <= LBOUND_5 + count;
                        RX6_wea <= '0';
                        RX6_addra <= LBOUND_6 + count;
                        RX7_wea <= '0';
                        RX7_addra <= LBOUND_7 + count;
                        RX8_wea <= '0';
                        RX8_addra <= LBOUND_8 + count;

                        SP1a_wea <= '0';
                        SP1b_wea <= '0';
                        SP1a_addra <= RX1_douta(46 DOWNTO
40);
                        SP1b_addra <= RX1_douta(38 DOWNTO
32);
                        SP2a_wea <= '0';
```

```
                                SP2b_wea <= '0';
                                SP2a_addra <= RX2_douta(46 DOWNTO
40);
                                SP2b_addra <= RX2_douta(38 DOWNTO
32);
                                SP3a_wea <= '0';
                                SP3b_wea <= '0';
                                SP3a_addra <= RX3_douta(46 DOWNTO
40);
                                SP3b_addra <= RX3_douta(38 DOWNTO
32);
                                SP4a_wea <= '0';
                                SP4b_wea <= '0';
                                SP4a_addra <= RX4_douta(46 DOWNTO
40);
                                SP4b_addra <= RX4_douta(38 DOWNTO
32);
                                SP5a_wea <= '0';
                                SP5b_wea <= '0';
                                SP5a_addra <= RX5_douta(46 DOWNTO
40);
                                SP5b_addra <= RX5_douta(38 DOWNTO
32);
                                SP6a_wea <= '0';
                                SP6b_wea <= '0';
                                SP6a_addra <= RX6_douta(46 DOWNTO
40);
                                SP6b_addra <= RX6_douta(38 DOWNTO
32);
                                SP7a_wea <= '0';
                                SP7b_wea <= '0';
                                SP7a_addra <= RX7_douta(46 DOWNTO
40);
                                SP7b_addra <= RX7_douta(38 DOWNTO
32);
                                SP8a_wea <= '0';
                                SP8b_wea <= '0';
                                SP8a_addra <= RX8_douta(46 DOWNTO
40);
                                SP8b_addra <= RX8_douta(38 DOWNTO
32);

                                v_PC1_POP1 := SP1a_douta;
                                v_PC1_POP2 := SP1b_douta;
                                v_PC1_RX := RX1_douta;
                                v_PC2_POP1 := SP2a_douta;
                                v_PC2_POP2 := SP2b_douta;
                                v_PC2_RX := RX2_douta;
                                v_PC3_POP1 := SP3a_douta;
                                v_PC3_POP2 := SP3b_douta;
                                v_PC3_RX := RX3_douta;
                                v_PC4_POP1 := SP4a_douta;
                                v_PC4_POP2 := SP4b_douta;
                                v_PC4_RX := RX4_douta;
                                v_PC5_POP1 := SP5a_douta;
                                v_PC5_POP2 := SP5b_douta;
                                v_PC5_RX := RX5_douta;
```

```
      v_PC6_POP1 := SP6a_douta;
      v_PC6_POP2 := SP6b_douta;
      v_PC6_RX := RX6_douta;
      v_PC7_POP1 := SP7a_douta;
      v_PC7_POP2 := SP7b_douta;
      v_PC7_RX := RX7_douta;
      v_PC8_POP1 := SP8a_douta;
      v_PC8_POP2 := SP8b_douta;
      v_PC8_RX := RX8_douta;
      state2 := state2 + 1;
ELSIF (state2 = "00000011") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00000100") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00000101") THEN
      we <= '0';
      addr <= X"00";

      P1_wea <= '1';
      P1_addra <= LBOUND_1 + count;
      P1_dina <= PC1_PROP;
      P2_wea <= '1';
      P2_addra <= LBOUND_2 + count;
      P2_dina <= PC2_PROP;
      P3_wea <= '1';
      P3_addra <= LBOUND_3 + count;
      P3_dina <= PC3_PROP;
      P4_wea <= '1';
      P4_addra <= LBOUND_4 + count;
      P4_dina <= PC4_PROP;
      P5_wea <= '1';
      P5_addra <= LBOUND_5 + count;
      P5_dina <= PC5_PROP;
      P6_wea <= '1';
      P6_addra <= LBOUND_6 + count;
      P6_dina <= PC6_PROP;
      P7_wea <= '1';
      P7_addra <= LBOUND_7 + count;
      P7_dina <= PC7_PROP;
      P8_wea <= '1';
      P8_addra <= LBOUND_8 + count;
      P8_dina <= PC8_PROP;
      IF (count = "000000") THEN
            v_PSUM1_1 := PC1_PROP;
            v_PSUM2_1 := PC2_PROP;
            v_PSUM3_1 := PC3_PROP;
            v_PSUM4_1 := PC4_PROP;
            v_PSUM5_1 := PC5_PROP;
            v_PSUM6_1 := PC6_PROP;
            v_PSUM7_1 := PC7_PROP;
            v_PSUM8_1 := PC8_PROP;
```

```
                                        state2 := "00000000";
                                        count := count + 1;
                                    ELSIF (count < "000100") THEN
                                        v_PSUM1_1 := v_PSUM1_1 +
PC1_PROP;
                                        v_PSUM2_1 := v_PSUM2_1 +
PC2_PROP;
                                        v_PSUM3_1 := v_PSUM3_1 +
PC3_PROP;
                                        v_PSUM4_1 := v_PSUM4_1 +
PC4_PROP;
                                        v_PSUM5_1 := v_PSUM5_1 +
PC5_PROP;
                                        v_PSUM6_1 := v_PSUM6_1 +
PC6_PROP;
                                        v_PSUM7_1 := v_PSUM7_1 +
PC7_PROP;
                                        v_PSUM8_1 := v_PSUM8_1 +
PC8_PROP;
                                        state2 := "00000000";
                                        count := count + 1;
                                    ELSIF (count = "000100") THEN
                                        v_PSUM1_2 := v_PSUM1_1 +
PC1_PROP;
                                        v_PSUM2_2 := v_PSUM2_1 +
PC2_PROP;
                                        v_PSUM3_2 := v_PSUM3_1 +
PC3_PROP;
                                        v_PSUM4_2 := v_PSUM4_1 +
PC4_PROP;
                                        v_PSUM5_2 := v_PSUM5_1 +
PC5_PROP;
                                        v_PSUM6_2 := v_PSUM6_1 +
PC6_PROP;
                                        v_PSUM7_2 := v_PSUM7_1 +
PC7_PROP;
                                        v_PSUM8_2 := v_PSUM8_1 +
PC8_PROP;
                                        state2 := "00000000";
                                        count := count + 1;
                                    ELSIF (count < "000111") THEN
                                        v_PSUM1_2 := v_PSUM1_2 +
PC1_PROP;
                                        v_PSUM2_2 := v_PSUM2_2 +
PC2_PROP;
                                        v_PSUM3_2 := v_PSUM3_2 +
PC3_PROP;
                                        v_PSUM4_2 := v_PSUM4_2 +
PC4_PROP;
                                        v_PSUM5_2 := v_PSUM5_2 +
PC5_PROP;
                                        v_PSUM6_2 := v_PSUM6_2 +
PC6_PROP;
                                        v_PSUM7_2 := v_PSUM7_2 +
PC7_PROP;
                                        v_PSUM8_2 := v_PSUM8_2 +
PC8_PROP;
```

```
                                    state2 := "00000000";
                                    count := count + 1;
                              ELSE
                                    v_PSUM1_2 := v_PSUM1_2 +
PC1_PROP;
                                    v_PSUM2_2 := v_PSUM2_2 +
PC2_PROP;
                                    v_PSUM3_2 := v_PSUM3_2 +
PC3_PROP;
                                    v_PSUM4_2 := v_PSUM4_2 +
PC4_PROP;
                                    v_PSUM5_2 := v_PSUM5_2 +
PC5_PROP;
                                    v_PSUM6_2 := v_PSUM6_2 +
PC6_PROP;
                                    v_PSUM7_2 := v_PSUM7_2 +
PC7_PROP;
                                    v_PSUM8_2 := v_PSUM8_2 +
PC8_PROP;
                                    state2 := state2 + 1;
                                    count := "000000";
                              END IF;

                        ELSIF (state2 = "00000110") THEN
                              P1_wea <= '0';
                              P2_wea <= '0';
                              P3_wea <= '0';
                              P4_wea <= '0';
                              P5_wea <= '0';
                              P6_wea <= '0';
                              P7_wea <= '0';
                              P8_wea <= '0';
                              we <= '0';
                              addr <= X"00";
                              state2 := state2 + 1;
                        ELSIF (state2 = "00000111") THEN
                              we <= '1';
                              addr <= index;
                              din(63 DOWNTO 32) <= TPROP8(31
DOWNTO 0);
                              din(31 DOWNTO 0) <= s_ERV;
                              state2 := state2 + 1;

                        ELSIF (state2 = "00001000") THEN
                              we <= '0';
                              addr <= X"00";
                              index := index + 1;
                              state2 := state2 + 1;

                        ELSIF (state2 = "00001001") THEN
                              we <= '0';
                              addr <= X"00";
                              state2 := state2 + 1;

                        ELSIF (state2 = "00001010") THEN
                              we <= '0';
                              addr <= X"00";
```

```
      state2 := state2 + 1;

ELSIF (state2 = "00001011") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00001100") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00001101") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00001110") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00001111") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00010000") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00010001") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00010010") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00010011") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00010100") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;

ELSIF (state2 = "00010101") THEN
      we <= '0';
      addr <= X"00";
      state2 := state2 + 1;
```

```
                        ELSIF (state2 = "00010110") THEN
                              we <= '0';
                              addr <= X"00";
                              state2 := state2 + 1;

                        ELSIF (state2 = "00010111") THEN
                              we <= '0';
                              addr <= X"00";
                              state2 := state2 + 1;

                        ELSIF (state2 = "00011000") THEN
                              we <= '0';
                              addr <= X"00";
                              state2 := state2 + 1;

                        ELSIF (state2 = "00011001") THEN
                              we <= '0';
                              addr <= X"00";
                              state2 := state2 + 1;

                        ELSIF (state2 = "00011010") THEN
                              we <= '0';
                              addr <= X"00";
                              state2 := state2 + 1;

                        ELSIF (state2 = "00011011") THEN
                              we <= '1';
                              addr <= index;
                              din(63 DOWNTO 32) <= product(63
DOWNTO 32);
                              din(31 DOWNTO 6) <=
"00000000000000000000000000";
                              din(5 DOWNTO 0) <= s_rxselect;
                              RXUS_wea <= '0';
                              RXUS_addra <= s_rxselect;
                              v_RX := s_rxselect;

                              state2 := state2 + 1;

                        ELSIF (state2 = "00011100") THEN
                              we <= '0';
                              addr <= X"00";

                              s_lfsr_reset <= '0';
                              s_lfsr_enable <= '1';
                              RXUS_wea <= '0';
                              RXUS_addra <= v_RX;

                              SPUS1_wea <= '0';
                              SPUS2_wea <= '0';
                              SPUS3_wea <= '0';
                              SPUS4_wea <= '0';

                              SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                              SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
```

```
                              SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS4_addra <= RXUS_douta(22 DOWNTO
16);

                              state2 := state2 + 1;

                        ELSIF (state2 = "00011101") THEN
                              we <= '0';
                              addr <= X"00";
                              s_lfsr_reset <= '0';
                              s_lfsr_enable <= '0';
                              RXUS_wea <= '0';
                              RXUS_addra <= v_RX;

                              SPUS1_wea <= '0';
                              SPUS2_wea <= '0';
                              SPUS3_wea <= '0';
                              SPUS4_wea <= '0';

                              SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                              SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                              SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS4_addra <= RXUS_douta(22 DOWNTO
16);

                              IF (RXUS_douta(46 DOWNTO 40) =
"1111111") THEN
                                    state2 := state2 + 2;
                              ELSE
                                    v_R1V := SPUS1_douta - 1;
                                    state2 := state2 + 1;
                              END IF;

                        ELSIF (state2 = "00011110") THEN
                              we <= '0';
                              addr <= X"00";
                              RXUS_wea <= '0';
                              RXUS_addra <= v_RX;

                              SPUS1_wea <= '1';
                              SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                              SPUS1_dina <= v_R1V;
                              SPUS2_wea <= '1';
                              SPUS2_addra <= RXUS_douta(46 DOWNTO
40);
                              SPUS2_dina <= v_R1V;
                              SPUS3_wea <= '1';
                              SPUS3_addra <= RXUS_douta(46 DOWNTO
40);
                              SPUS3_dina <= v_R1V;
                              SPUS4_wea <= '1';
```

```
                                SPUS4_addra <= RXUS_douta(46 DOWNTO
40);
                                SPUS4_dina <= v_R1V;
                                SP1a_wea <= '1';
                                SP1a_addra <= RXUS_douta(46 DOWNTO
40);
                                SP1a_dina <= v_R1V;
                                SP1b_wea <= '1';
                                SP1b_addra <= RXUS_douta(46 DOWNTO
40);
                                SP1b_dina <= v_R1V;
                                SP2a_wea <= '1';
                                SP2a_addra <= RXUS_douta(46 DOWNTO
40);
                                SP2a_dina <= v_R1V;
                                SP2b_wea <= '1';
                                SP2b_addra <= RXUS_douta(46 DOWNTO
40);
                                SP2b_dina <= v_R1V;
                                SP3a_wea <= '1';
                                SP3a_addra <= RXUS_douta(46 DOWNTO
40);
                                SP3a_dina <= v_R1V;
                                SP3b_wea <= '1';
                                SP3b_addra <= RXUS_douta(46 DOWNTO
40);
                                SP3b_dina <= v_R1V;
                                SP4a_wea <= '1';
                                SP4a_addra <= RXUS_douta(46 DOWNTO
40);
                                SP4a_dina <= v_R1V;
                                SP4b_wea <= '1';
                                SP4b_addra <= RXUS_douta(46 DOWNTO
40);
                                SP4b_dina <= v_R1V;
                                SP5a_wea <= '1';
                                SP5a_addra <= RXUS_douta(46 DOWNTO
40);
                                SP5a_dina <= v_R1V;
                                SP5b_wea <= '1';
                                SP5b_addra <= RXUS_douta(46 DOWNTO
40);
                                SP5b_dina <= v_R1V;
                                SP6a_wea <= '1';
                                SP6a_addra <= RXUS_douta(46 DOWNTO
40);
                                SP6a_dina <= v_R1V;
                                SP6b_wea <= '1';
                                SP6b_addra <= RXUS_douta(46 DOWNTO
40);
                                SP6b_dina <= v_R1V;
                                SP7a_wea <= '1';
                                SP7a_addra <= RXUS_douta(46 DOWNTO
40);
                                SP7a_dina <= v_R1V;
                                SP7b_wea <= '1';
```

```
                        SP7b_addra <= RXUS_douta(46 DOWNTO
40);
                        SP7b_dina <= v_R1V;
                        SP8a_wea <= '1';
                        SP8a_addra <= RXUS_douta(46 DOWNTO
40);
                        SP8a_dina <= v_R1V;
                        SP8b_wea <= '1';
                        SP8b_addra <= RXUS_douta(46 DOWNTO
40);
                        SP8b_dina <= v_R1V;

                        state2 := state2 + 1;

                    ELSIF (state2 = "00011111") THEN
                        we <= '0';
                        addr <= X"00";
                        RXUS_wea <= '0';
                        RXUS_addra <= v_RX;

                        SPUS1_wea <= '0';
                        SPUS2_wea <= '0';
                        SPUS3_wea <= '0';
                        SPUS4_wea <= '0';
                        SP1a_wea <= '0';
                        SP1b_wea <= '0';
                        SP2a_wea <= '0';
                        SP2b_wea <= '0';
                        SP3a_wea <= '0';
                        SP3b_wea <= '0';
                        SP4a_wea <= '0';
                        SP4b_wea <= '0';
                        SP5a_wea <= '0';
                        SP5b_wea <= '0';
                        SP6a_wea <= '0';
                        SP6b_wea <= '0';
                        SP7a_wea <= '0';
                        SP7b_wea <= '0';
                        SP8a_wea <= '0';
                        SP8b_wea <= '0';

                        SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                        SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                        SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                        SPUS4_addra <= RXUS_douta(22 DOWNTO
16);

                        state2 := state2 + 1;

                    ELSIF (state2 = "00100000") THEN
                        we <= '0';
                        addr <= X"00";
                        RXUS_wea <= '0';
                        RXUS_addra <= v_RX;
```

```
                                    SPUS1_wea <= '0';
                                    SPUS2_wea <= '0';
                                    SPUS3_wea <= '0';
                                    SPUS4_wea <= '0';

                                    SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                                    SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                                    SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                                    SPUS4_addra <= RXUS_douta(22 DOWNTO
16);

                                    IF (RXUS_douta(38 DOWNTO 32) =
"1111111") THEN
                                          state2 := state2 + 2;
                                    ELSE
                                          v_R2V := SPUS2_douta - 1;
                                          state2 := state2 + 1;
                                    END IF;

                              ELSIF (state2 = "00100001") THEN
                                    we <= '0';
                                    addr <= X"00";
                                    RXUS_wea <= '0';
                                    RXUS_addra <= v_RX;

                                    SPUS1_wea <= '1';
                                    SPUS1_addra <= RXUS_douta(38 DOWNTO
32);
                                    SPUS1_dina <= v_R2V;
                                    SPUS2_wea <= '1';
                                    SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                                    SPUS2_dina <= v_R2V;
                                    SPUS3_wea <= '1';
                                    SPUS3_addra <= RXUS_douta(38 DOWNTO
32);
                                    SPUS3_dina <= v_R2V;
                                    SPUS4_wea <= '1';
                                    SPUS4_addra <= RXUS_douta(38 DOWNTO
32);
                                    SPUS4_dina <= v_R2V;
                                    SP1a_wea <= '1';
                                    SP1a_addra <= RXUS_douta(38 DOWNTO
32);
                                    SP1a_dina <= v_R2V;
                                    SP1b_wea <= '1';
                                    SP1b_addra <= RXUS_douta(38 DOWNTO
32);
                                    SP1b_dina <= v_R2V;
                                    SP2a_wea <= '1';
                                    SP2a_addra <= RXUS_douta(38 DOWNTO
32);
                                    SP2a_dina <= v_R2V;
```

```
                        SP2b_wea <= '1';
                        SP2b_addra <= RXUS_douta(38 DOWNTO
32);
                        SP2b_dina <= v_R2V;
                        SP3a_wea <= '1';
                        SP3a_addra <= RXUS_douta(38 DOWNTO
32);
                        SP3a_dina <= v_R2V;
                        SP3b_wea <= '1';
                        SP3b_addra <= RXUS_douta(38 DOWNTO
32);
                        SP3b_dina <= v_R2V;
                        SP4a_wea <= '1';
                        SP4a_addra <= RXUS_douta(38 DOWNTO
32);
                        SP4a_dina <= v_R2V;
                        SP4b_wea <= '1';
                        SP4b_addra <= RXUS_douta(38 DOWNTO
32);
                        SP4b_dina <= v_R2V;
                        SP5a_wea <= '1';
                        SP5a_addra <= RXUS_douta(38 DOWNTO
32);
                        SP5a_dina <= v_R2V;
                        SP5b_wea <= '1';
                        SP5b_addra <= RXUS_douta(38 DOWNTO
32);
                        SP5b_dina <= v_R2V;
                        SP6a_wea <= '1';
                        SP6a_addra <= RXUS_douta(38 DOWNTO
32);
                        SP6a_dina <= v_R2V;
                        SP6b_wea <= '1';
                        SP6b_addra <= RXUS_douta(38 DOWNTO
32);
                        SP6b_dina <= v_R2V;
                        SP7a_wea <= '1';
                        SP7a_addra <= RXUS_douta(38 DOWNTO
32);
                        SP7a_dina <= v_R2V;
                        SP7b_wea <= '1';
                        SP7b_addra <= RXUS_douta(38 DOWNTO
32);
                        SP7b_dina <= v_R2V;
                        SP8a_wea <= '1';
                        SP8a_addra <= RXUS_douta(38 DOWNTO
32);
                        SP8a_dina <= v_R2V;
                        SP8b_wea <= '1';
                        SP8b_addra <= RXUS_douta(38 DOWNTO
32);
                        SP8b_dina <= v_R2V;

                        state2 := state2 + 1;

                    ELSIF (state2 = "00100010") THEN
                        we <= '0';
```

```
                addr <= X"00";
                RXUS_wea <= '0';
                RXUS_addra <= v_RX;

                SPUS1_wea <= '0';
                SPUS2_wea <= '0';
                SPUS3_wea <= '0';
                SPUS4_wea <= '0';
                SP1a_wea <= '0';
                SP1b_wea <= '0';
                SP2a_wea <= '0';
                SP2b_wea <= '0';
                SP3a_wea <= '0';
                SP3b_wea <= '0';
                SP4a_wea <= '0';
                SP4b_wea <= '0';
                SP5a_wea <= '0';
                SP5b_wea <= '0';
                SP6a_wea <= '0';
                SP6b_wea <= '0';
                SP7a_wea <= '0';
                SP7b_wea <= '0';
                SP8a_wea <= '0';
                SP8b_wea <= '0';

                SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                SPUS4_addra <= RXUS_douta(22 DOWNTO
16);

                state2 := state2 + 1;

            ELSIF (state2 = "00100011") THEN
                we <= '0';
                addr <= X"00";
                RXUS_wea <= '0';
                RXUS_addra <= v_RX;

                SPUS1_wea <= '0';
                SPUS2_wea <= '0';
                SPUS3_wea <= '0';
                SPUS4_wea <= '0';

                SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                SPUS4_addra <= RXUS_douta(22 DOWNTO
16);
```

```
                              IF (RXUS_douta(30 DOWNTO 24) =
"1111111") THEN
                                    state2 := state2 + 2;
                              ELSE
                                    v_P1V := SPUS3_douta + 1;
                                    state2 := state2 + 1;
                              END IF;

                        ELSIF (state2 = "00100100") THEN
                              we <= '0';
                              addr <= X"00";
                              RXUS_wea <= '0';
                              RXUS_addra <= v_RX;

                              SPUS1_wea <= '1';
                              SPUS1_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS1_dina <= v_P1V;
                              SPUS2_wea <= '1';
                              SPUS2_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS2_dina <= v_P1V;
                              SPUS3_wea <= '1';
                              SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS3_dina <= v_P1V;
                              SPUS4_wea <= '1';
                              SPUS4_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS4_dina <= v_P1V;
                              SP1a_wea <= '1';
                              SP1a_addra <= RXUS_douta(30 DOWNTO
24);
                              SP1a_dina <= v_P1V;
                              SP1b_wea <= '1';
                              SP1b_addra <= RXUS_douta(30 DOWNTO
24);
                              SP1b_dina <= v_P1V;
                              SP2a_wea <= '1';
                              SP2a_addra <= RXUS_douta(30 DOWNTO
24);
                              SP2a_dina <= v_P1V;
                              SP2b_wea <= '1';
                              SP2b_addra <= RXUS_douta(30 DOWNTO
24);
                              SP2b_dina <= v_P1V;
                              SP3a_wea <= '1';
                              SP3a_addra <= RXUS_douta(30 DOWNTO
24);
                              SP3a_dina <= v_P1V;
                              SP3b_wea <= '1';
                              SP3b_addra <= RXUS_douta(30 DOWNTO
24);
                              SP3b_dina <= v_P1V;
                              SP4a_wea <= '1';
                              SP4a_addra <= RXUS_douta(30 DOWNTO
24);
```

```
                        SP4a_dina <= v_P1V;
                        SP4b_wea <= '1';
                        SP4b_addra <= RXUS_douta(30 DOWNTO
24);
                        SP4b_dina <= v_P1V;
                        SP5a_wea <= '1';
                        SP5a_addra <= RXUS_douta(30 DOWNTO
24);
                        SP5a_dina <= v_P1V;
                        SP5b_wea <= '1';
                        SP5b_addra <= RXUS_douta(30 DOWNTO
24);
                        SP5b_dina <= v_P1V;
                        SP6a_wea <= '1';
                        SP6a_addra <= RXUS_douta(30 DOWNTO
24);
                        SP6a_dina <= v_P1V;
                        SP6b_wea <= '1';
                        SP6b_addra <= RXUS_douta(30 DOWNTO
24);
                        SP6b_dina <= v_P1V;
                        SP7a_wea <= '1';
                        SP7a_addra <= RXUS_douta(30 DOWNTO
24);
                        SP7a_dina <= v_P1V;
                        SP7b_wea <= '1';
                        SP7b_addra <= RXUS_douta(30 DOWNTO
24);
                        SP7b_dina <= v_P1V;
                        SP8a_wea <= '1';
                        SP8a_addra <= RXUS_douta(30 DOWNTO
24);
                        SP8a_dina <= v_P1V;
                        SP8b_wea <= '1';
                        SP8b_addra <= RXUS_douta(30 DOWNTO
24);
                        SP8b_dina <= v_P1V;

                        state2 := state2 + 1;

                    ELSIF (state2 = "00100101") THEN
                        we <= '0';
                        addr <= X"00";
                        RXUS_wea <= '0';
                        RXUS_addra <= v_RX;

                        SPUS1_wea <= '0';
                        SPUS2_wea <= '0';
                        SPUS3_wea <= '0';
                        SPUS4_wea <= '0';
                        SP1a_wea <= '0';
                        SP1b_wea <= '0';
                        SP2a_wea <= '0';
                        SP2b_wea <= '0';
                        SP3a_wea <= '0';
                        SP3b_wea <= '0';
                        SP4a_wea <= '0';
```

```
                              SP4b_wea <= '0';
                              SP5a_wea <= '0';
                              SP5b_wea <= '0';
                              SP6a_wea <= '0';
                              SP6b_wea <= '0';
                              SP7a_wea <= '0';
                              SP7b_wea <= '0';
                              SP8a_wea <= '0';
                              SP8b_wea <= '0';

                              SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                              SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                              SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS4_addra <= RXUS_douta(22 DOWNTO
16);


                              state2 := state2 + 1;

                        ELSIF (state2 = "00100110") THEN
                              we <= '0';
                              addr <= X"00";
                              RXUS_wea <= '0';
                              RXUS_addra <= v_RX;

                              SPUS1_wea <= '0';
                              SPUS2_wea <= '0';
                              SPUS3_wea <= '0';
                              SPUS4_wea <= '0';

                              SPUS1_addra <= RXUS_douta(46 DOWNTO
40);
                              SPUS2_addra <= RXUS_douta(38 DOWNTO
32);
                              SPUS3_addra <= RXUS_douta(30 DOWNTO
24);
                              SPUS4_addra <= RXUS_douta(22 DOWNTO
16);


                              IF (RXUS_douta(22 DOWNTO 16) =
"1111111") THEN
                                    state2 := X"00";
                                    index := index + 1;
                              ELSE
                                    v_P2V := SPUS4_douta - 1;
                                    state2 := state2 + 1;
                              END IF;

                        ELSIF (state2 = "00100111") THEN
                              we <= '0';
                              addr <= X"00";
                              RXUS_wea <= '0';
                              RXUS_addra <= v_RX;

                              SPUS1_wea <= '1';
```

```
                                SPUS1_addra <= RXUS_douta(22 DOWNTO
16);
                                SPUS1_dina <= v_P2V;
                                SPUS2_wea <= '1';
                                SPUS2_addra <= RXUS_douta(22 DOWNTO
16);
                                SPUS2_dina <= v_P2V;
                                SPUS3_wea <= '1';
                                SPUS3_addra <= RXUS_douta(22 DOWNTO
16);
                                SPUS3_dina <= v_P2V;
                                SPUS4_wea <= '1';
                                SPUS4_addra <= RXUS_douta(22 DOWNTO
16);
                                SPUS4_dina <= v_P2V;
                                SP1a_wea <= '1';
                                SP1a_addra <= RXUS_douta(22 DOWNTO
16);
                                SP1a_dina <= v_P2V;
                                SP1b_wea <= '1';
                                SP1b_addra <= RXUS_douta(22 DOWNTO
16);
                                SP1b_dina <= v_P2V;
                                SP2a_wea <= '1';
                                SP2a_addra <= RXUS_douta(22 DOWNTO
16);
                                SP2a_dina <= v_P2V;
                                SP2b_wea <= '1';
                                SP2b_addra <= RXUS_douta(22 DOWNTO
16);
                                SP2b_dina <= v_P2V;
                                SP3a_wea <= '1';
                                SP3a_addra <= RXUS_douta(22 DOWNTO
16);
                                SP3a_dina <= v_P2V;
                                SP3b_wea <= '1';
                                SP3b_addra <= RXUS_douta(22 DOWNTO
16);
                                SP3b_dina <= v_P2V;
                                SP4a_wea <= '1';
                                SP4a_addra <= RXUS_douta(22 DOWNTO
16);
                                SP4a_dina <= v_P2V;
                                SP4b_wea <= '1';
                                SP4b_addra <= RXUS_douta(22 DOWNTO
16);
                                SP4b_dina <= v_P2V;
                                SP5a_wea <= '1';
                                SP5a_addra <= RXUS_douta(22 DOWNTO
16);
                                SP5a_dina <= v_P2V;
                                SP5b_wea <= '1';
                                SP5b_addra <= RXUS_douta(22 DOWNTO
16);
                                SP5b_dina <= v_P2V;
                                SP6a_wea <= '1';
```

```
                                    SP6a_addra <= RXUS_douta(22 DOWNTO
16);
                                    SP6a_dina <= v_P2V;
                                    SP6b_wea <= '1';
                                    SP6b_addra <= RXUS_douta(22 DOWNTO
16);
                                    SP6b_dina <= v_P2V;
                                    SP7a_wea <= '1';
                                    SP7a_addra <= RXUS_douta(22 DOWNTO
16);
                                    SP7a_dina <= v_P2V;
                                    SP7b_wea <= '1';
                                    SP7b_addra <= RXUS_douta(22 DOWNTO
16);
                                    SP7b_dina <= v_P2V;
                                    SP8a_wea <= '1';
                                    SP8a_addra <= RXUS_douta(22 DOWNTO
16);
                                    SP8a_dina <= v_P2V;
                                    SP8b_wea <= '1';
                                    SP8b_addra <= RXUS_douta(22 DOWNTO
16);
                                    SP8b_dina <= v_P2V;

                                    index := index + 1;
                                    state2 := X"00";
                              END IF;
                        ELSE
                              we <= '0';
                              addr <= X"00";

                              --JUST ADDED
                              SPUS1_wea <= '0';
                              SPUS2_wea <= '0';
                              SPUS3_wea <= '0';
                              SPUS4_wea <= '0';
                              SP1a_wea <= '0';
                              SP1b_wea <= '0';
                              SP2a_wea <= '0';
                              SP2b_wea <= '0';
                              SP3a_wea <= '0';
                              SP3b_wea <= '0';
                              SP4a_wea <= '0';
                              SP4b_wea <= '0';
                              SP5a_wea <= '0';
                              SP5b_wea <= '0';
                              SP6a_wea <= '0';
                              SP6b_wea <= '0';
                              SP7a_wea <= '0';
                              SP7b_wea <= '0';
                              SP8a_wea <= '0';
                              SP8b_wea <= '0';

                              looping := '0';
                              state := state + 1;
                        END IF;
```

```
-- NO-OP
ELSIF (dout(63 DOWNTO 59) = "00000") THEN
      we <= '0';
      addr <= X"00";
      state := "00000000";

-- SETTING SPECIES POPULATIONS
ELSIF (dout(63 DOWNTO 59) = "00001") THEN
      we <= '0';
      addr <= X"00";
      SPUS1_wea <= '1';
      SPUS1_addra <= dout(57 DOWNTO 51);
      SPUS1_dina <= dout(15 DOWNTO 0);
      SPUS2_wea <= '1';
      SPUS2_addra <= dout(57 DOWNTO 51);
      SPUS2_dina <= dout(15 DOWNTO 0);
      SPUS3_wea <= '1';
      SPUS3_addra <= dout(57 DOWNTO 51);
      SPUS3_dina <= dout(15 DOWNTO 0);
      SPUS4_wea <= '1';
      SPUS4_addra <= dout(57 DOWNTO 51);
      SPUS4_dina <= dout(15 DOWNTO 0);
      SP1a_wea <= '1';
      SP1a_addra <= dout(57 DOWNTO 51);
      SP1a_dina <= dout(15 DOWNTO 0);
      SP1b_wea <= '1';
      SP1b_addra <= dout(57 DOWNTO 51);
      SP1b_dina <= dout(15 DOWNTO 0);
      SP2a_wea <= '1';
      SP2a_addra <= dout(57 DOWNTO 51);
      SP2a_dina <= dout(15 DOWNTO 0);
      SP2b_wea <= '1';
      SP2b_addra <= dout(57 DOWNTO 51);
      SP2b_dina <= dout(15 DOWNTO 0);
      SP3a_wea <= '1';
      SP3a_addra <= dout(57 DOWNTO 51);
      SP3a_dina <= dout(15 DOWNTO 0);
      SP3b_wea <= '1';
      SP3b_addra <= dout(57 DOWNTO 51);
      SP3b_dina <= dout(15 DOWNTO 0);
      SP4a_wea <= '1';
      SP4a_addra <= dout(57 DOWNTO 51);
      SP4a_dina <= dout(15 DOWNTO 0);
      SP4b_wea <= '1';
      SP4b_addra <= dout(57 DOWNTO 51);
      SP4b_dina <= dout(15 DOWNTO 0);
      SP5a_wea <= '1';
      SP5a_addra <= dout(57 DOWNTO 51);
      SP5a_dina <= dout(15 DOWNTO 0);
      SP5b_wea <= '1';
      SP5b_addra <= dout(57 DOWNTO 51);
      SP5b_dina <= dout(15 DOWNTO 0);
      SP6a_wea <= '1';
      SP6a_addra <= dout(57 DOWNTO 51);
      SP6a_dina <= dout(15 DOWNTO 0);
      SP6b_wea <= '1';
      SP6b_addra <= dout(57 DOWNTO 51);
```

```
            SP6b_dina <= dout(15 DOWNTO 0);
            SP7a_wea <= '1';
            SP7a_addra <= dout(57 DOWNTO 51);
            SP7a_dina <= dout(15 DOWNTO 0);
            SP7b_wea <= '1';
            SP7b_addra <= dout(57 DOWNTO 51);
            SP7b_dina <= dout(15 DOWNTO 0);
            SP8a_wea <= '1';
            SP8a_addra <= dout(57 DOWNTO 51);
            SP8a_dina <= dout(15 DOWNTO 0);
            SP8b_wea <= '1';
            SP8b_addra <= dout(57 DOWNTO 51);
            SP8b_dina <= dout(15 DOWNTO 0);

            state := state + 1;

      -- READING A SPECIES POPULATION
      ELSIF (dout(63 DOWNTO 59) = "00010") THEN
            IF (state2 = "00000000") THEN
                  state2 := state2 + 1;
                  we <= '0';
                  addr <= X"00";

                  SP1a_wea <= '0';
                  SP1a_addra <= dout(57 DOWNTO 51);
            ELSE
                  we <= '1';
                  addr <= X"01";
                  SP1a_wea <= '0';
                  SP1a_addra <= dout(57 DOWNTO 51);
                  din(63 DOWNTO 16) <= X"000000000000";
                  din(15 DOWNTO 0) <= SP1a_douta;
                  state2 := "00000000";
                  state := state + 1;
            END IF;

      -- SETTING A REACTION EQUATION
      ELSIF (dout(63 DOWNTO 59) = "00011") THEN
            we <= '0';
            addr <= X"00";
            RX1_wea <= '1';
            RX1_addra <= dout(56 DOWNTO 51);
            RX1_dina <= dout(47 DOWNTO 0);
            RX2_wea <= '1';
            RX2_addra <= dout(56 DOWNTO 51);
            RX2_dina <= dout(47 DOWNTO 0);
            RX3_wea <= '1';
            RX3_addra <= dout(56 DOWNTO 51);
            RX3_dina <= dout(47 DOWNTO 0);
            RX4_wea <= '1';
            RX4_addra <= dout(56 DOWNTO 51);
            RX4_dina <= dout(47 DOWNTO 0);
            RX5_wea <= '1';
            RX5_addra <= dout(56 DOWNTO 51);
            RX5_dina <= dout(47 DOWNTO 0);
            RX6_wea <= '1';
            RX6_addra <= dout(56 DOWNTO 51);
```

```
                        RX6_dina <= dout(47 DOWNTO 0);
                        RX7_wea <= '1';
                        RX7_addra <= dout(56 DOWNTO 51);
                        RX7_dina <= dout(47 DOWNTO 0);
                        RX8_wea <= '1';
                        RX8_addra <= dout(56 DOWNTO 51);
                        RX8_dina <= dout(47 DOWNTO 0);
                        RXUS_wea <= '1';
                        RXUS_addra <= dout(56 DOWNTO 51);
                        RXUS_dina <= dout(47 DOWNTO 0);
                        state := state + 1;

                  -- READING A REACTION EQUATION
                  ELSIF (dout(63 DOWNTO 59) = "00100") THEN
                        IF (state2 = "00000000") THEN
                              state2 := state2 + 1;
                              we <= '0';
                              addr <= X"00";

                              RX1_wea <= '0';
                              RX1_addra <= dout(56 DOWNTO 51);
                        ELSE
                              we <= '1';
                              addr <= X"01";
                              RX1_wea <= '0';
                              RX1_addra <= dout(56 DOWNTO 51);
                              din(63 DOWNTO 48) <= X"0000";
                              din(47 DOWNTO 0) <= RX1_douta(47 DOWNTO
0);
                              state2 := "00000000";
                              state := state + 1;
                        END IF;

                  -- READING A PROPENSITY
                  ELSIF (dout(63 DOWNTO 59) = "00101") THEN
                        IF (state2 = "00000000") THEN
                              state2 := state2 + 1;
                              we <= '0';
                              addr <= X"00";

                              P1_wea <= '0';
                              P1_addra <= dout(56 DOWNTO 51);
                              P2_wea <= '0';
                              P2_addra <= dout(56 DOWNTO 51);
                              P3_wea <= '0';
                              P3_addra <= dout(56 DOWNTO 51);
                              P4_wea <= '0';
                              P4_addra <= dout(56 DOWNTO 51);
                              P5_wea <= '0';
                              P5_addra <= dout(56 DOWNTO 51);
                              P6_wea <= '0';
                              P6_addra <= dout(56 DOWNTO 51);
                              P7_wea <= '0';
                              P7_addra <= dout(56 DOWNTO 51);
                              P8_wea <= '0';
                              P8_addra <= dout(56 DOWNTO 51);
                        ELSE
```

```
                              we <= '1';
                              addr <= X"01";
                              IF (dout(56 DOWNTO 51) < X"08") THEN
                                    P1_wea <= '0';
                                    P1_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P1_douta(47
DOWNTO 0);
                              ELSIF (dout(56 DOWNTO 51) < X"10") THEN
                                    P2_wea <= '0';
                                    P2_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P2_douta(47
DOWNTO 0);
                              ELSIF (dout(56 DOWNTO 51) < X"18") THEN
                                    P3_wea <= '0';
                                    P3_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P3_douta(47
DOWNTO 0);
                              ELSIF (dout(56 DOWNTO 51) < X"20") THEN
                                    P4_wea <= '0';
                                    P4_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P4_douta(47
DOWNTO 0);
                              ELSIF (dout(56 DOWNTO 51) < X"28") THEN
                                    P5_wea <= '0';
                                    P5_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P5_douta(47
DOWNTO 0);
                              ELSIF (dout(56 DOWNTO 51) < X"30") THEN
                                    P6_wea <= '0';
                                    P6_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P6_douta(47
DOWNTO 0);
                              ELSIF (dout(56 DOWNTO 51) < X"38") THEN
                                    P7_wea <= '0';
                                    P7_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P7_douta(47
DOWNTO 0);
                              ELSE
                                    P8_wea <= '0';
                                    P8_addra <= dout(56 DOWNTO 51);
                                    din(63 DOWNTO 48) <= X"0000";
                                    din(47 DOWNTO 0) <= P8_douta(47
DOWNTO 0);
                              END IF;
                              state2 := "00000000";
                              state := state + 1;
                        END IF;

      --          -- READING A PARTIAL SUM
      --          ELSIF (dout(63 DOWNTO 59) = "00110") THEN
```

```
--                we <= '1';
--                addr <= X"01";
--                din(63 DOWNTO 48) <= X"0000";
--                CASE dout(58 DOWNTO 51) IS
--                WHEN X"11" =>
--                      din(47 DOWNTO 0) <= v_PSUM1_1;
--                WHEN X"12" =>
--                      din(47 DOWNTO 0) <= v_PSUM1_2;
--                WHEN X"21" =>
--                      din(47 DOWNTO 0) <= v_PSUM2_1;
--                WHEN X"22" =>
--                      din(47 DOWNTO 0) <= v_PSUM2_2;
--                WHEN X"31" =>
--                      din(47 DOWNTO 0) <= v_PSUM3_1;
--                WHEN X"32" =>
--                      din(47 DOWNTO 0) <= v_PSUM3_2;
--                WHEN X"41" =>
--                      din(47 DOWNTO 0) <= v_PSUM4_1;
--                WHEN X"42" =>
--                      din(47 DOWNTO 0) <= v_PSUM4_2;
--                WHEN X"51" =>
--                      din(47 DOWNTO 0) <= v_PSUM5_1;
--                WHEN X"52" =>
--                      din(47 DOWNTO 0) <= v_PSUM5_2;
--                WHEN X"61" =>
--                      din(47 DOWNTO 0) <= v_PSUM6_1;
--                WHEN X"62" =>
--                      din(47 DOWNTO 0) <= v_PSUM6_2;
--                WHEN X"71" =>
--                      din(47 DOWNTO 0) <= v_PSUM7_1;
--                WHEN X"72" =>
--                      din(47 DOWNTO 0) <= v_PSUM7_2;
--                WHEN X"81" =>
--                      din(47 DOWNTO 0) <= v_PSUM8_1;
--                WHEN X"82" =>
--                      din(47 DOWNTO 0) <= v_PSUM8_2;
--                WHEN OTHERS =>
--                      din(47 DOWNTO 0) <= TPROP8;
--                END CASE;
--                state := state + 1;

             -- SET SEED TO UNIFORM RANDOM NUMBER GENERATOR
             ELSIF (dout(63 DOWNTO 59) = "00111") THEN
                   we <= '0';
                   addr <= X"00";
                   CASE state2 IS
                   WHEN "00000000" =>
                         s_seed <= dout(31 DOWNTO 0);
                         s_lfsr_reset <= '1';
                         state2 := state2 + 1;
                   WHEN OTHERS =>
                         s_seed <= dout(31 DOWNTO 0);
                         s_lfsr_reset <= '1';
                         s_lfsr_enable <= '1';
                         state := state + 1;
                         state2 := "00000000";
                   END CASE;
```

```
--          -- READING UNIFORM RANDOM NUMBER
--          ELSIF (dout(63 DOWNTO 59) = "01000") THEN
--                we <= '1';
--                addr <= X"01";
--                din(63 DOWNTO 32) <= X"00000000";
--                din(31 DOWNTO 0) <= s_URV;
--                state := state + 1;

--          -- CALCULATE A NEW UNIFORM RANDOM NUMBER
--          ELSIF (dout(63 DOWNTO 59) = "01001") THEN
--                we <= '0';
--                addr <= X"00";
--                s_lfsr_reset <= '0';
--                s_lfsr_enable <= '1';
--                state := state + 1;

                -- READING PRODUCT
                ELSIF (dout(63 DOWNTO 59) = "01010") THEN
                      we <= '1';
                      addr <= X"01";
                      din(63 DOWNTO 48) <= X"0000";
                      din(47 DOWNTO 0) <= product(79 DOWNTO 32);
                      state := state + 1;

                -- READING SELECTED REACTION
                ELSIF (dout(63 DOWNTO 59) = "01011") THEN
                      we <= '1';
                      addr <= X"01";
                      din(63 DOWNTO 8) <= X"00000000000000";
                      din(7 DOWNTO 6) <= "00";
                      din(5 DOWNTO 0) <= s_rxselect;
                      state := state + 1;

--          -- READING EXPONENTIAL RANDOM NUMBER
--          ELSIF (dout(63 DOWNTO 59) = "01100") THEN
--                we <= '1';
--                addr <= X"01";
--                din(63 DOWNTO 32) <= s_ERV_URV;
--                din(31 DOWNTO 0) <= s_ERV;
--                state := state + 1;

                -- INITIALIZING PROPENSITY CALCULATORS
                ELSIF (dout(63 DOWNTO 59) = "01101") THEN
                      we <= '0';
                      addr <= X"00";
                      IF (state2 < "00000010") THEN
                            P1_wea <= '0';
                            P2_wea <= '0';
                            P3_wea <= '0';
                            P4_wea <= '0';
                            P5_wea <= '0';
                            P6_wea <= '0';
                            P7_wea <= '0';
                            P8_wea <= '0';

                            RX1_wea <= '0';
```

```
            RX1_addra <= LBOUND_1 + count;
            RX2_wea <= '0';
            RX2_addra <= LBOUND_2 + count;
            RX3_wea <= '0';
            RX3_addra <= LBOUND_3 + count;
            RX4_wea <= '0';
            RX4_addra <= LBOUND_4 + count;
            RX5_wea <= '0';
            RX5_addra <= LBOUND_5 + count;
            RX6_wea <= '0';
            RX6_addra <= LBOUND_6 + count;
            RX7_wea <= '0';
            RX7_addra <= LBOUND_7 + count;
            RX8_wea <= '0';
            RX8_addra <= LBOUND_8 + count;
            state2 := state2 + 1;

        ELSIF (state2 < "00000100") THEN
            RX1_wea <= '0';
            RX1_addra <= LBOUND_1 + count;
            RX2_wea <= '0';
            RX2_addra <= LBOUND_2 + count;
            RX3_wea <= '0';
            RX3_addra <= LBOUND_3 + count;
            RX4_wea <= '0';
            RX4_addra <= LBOUND_4 + count;
            RX5_wea <= '0';
            RX5_addra <= LBOUND_5 + count;
            RX6_wea <= '0';
            RX6_addra <= LBOUND_6 + count;
            RX7_wea <= '0';
            RX7_addra <= LBOUND_7 + count;
            RX8_wea <= '0';
            RX8_addra <= LBOUND_8 + count;

            SP1a_wea <= '0';
            SP1b_wea <= '0';
            SP1a_addra <= RX1_douta(46 DOWNTO 40);
            SP1b_addra <= RX1_douta(38 DOWNTO 32);
            SP2a_wea <= '0';
            SP2b_wea <= '0';
            SP2a_addra <= RX2_douta(46 DOWNTO 40);
            SP2b_addra <= RX2_douta(38 DOWNTO 32);
            SP3a_wea <= '0';
            SP3b_wea <= '0';
            SP3a_addra <= RX3_douta(46 DOWNTO 40);
            SP3b_addra <= RX3_douta(38 DOWNTO 32);
            SP4a_wea <= '0';
            SP4b_wea <= '0';
            SP4a_addra <= RX4_douta(46 DOWNTO 40);
            SP4b_addra <= RX4_douta(38 DOWNTO 32);
            SP5a_wea <= '0';
            SP5b_wea <= '0';
            SP5a_addra <= RX5_douta(46 DOWNTO 40);
            SP5b_addra <= RX5_douta(38 DOWNTO 32);
            SP6a_wea <= '0';
            SP6b_wea <= '0';
```

```
            SP6a_addra <= RX6_douta(46 DOWNTO 40);
            SP6b_addra <= RX6_douta(38 DOWNTO 32);
            SP7a_wea <= '0';
            SP7b_wea <= '0';
            SP7a_addra <= RX7_douta(46 DOWNTO 40);
            SP7b_addra <= RX7_douta(38 DOWNTO 32);
            SP8a_wea <= '0';
            SP8b_wea <= '0';
            SP8a_addra <= RX8_douta(46 DOWNTO 40);
            SP8b_addra <= RX8_douta(38 DOWNTO 32);
            state2 := state2 + 1;

        ELSIF (state2 = "00000100") THEN
            RX1_wea <= '0';
            RX1_addra <= LBOUND_1 + count;
            RX2_wea <= '0';
            RX2_addra <= LBOUND_2 + count;
            RX3_wea <= '0';
            RX3_addra <= LBOUND_3 + count;
            RX4_wea <= '0';
            RX4_addra <= LBOUND_4 + count;
            RX5_wea <= '0';
            RX5_addra <= LBOUND_5 + count;
            RX6_wea <= '0';
            RX6_addra <= LBOUND_6 + count;
            RX7_wea <= '0';
            RX7_addra <= LBOUND_7 + count;
            RX8_wea <= '0';
            RX8_addra <= LBOUND_8 + count;

            SP1a_wea <= '0';
            SP1b_wea <= '0';
            SP1a_addra <= RX1_douta(46 DOWNTO 40);
            SP1b_addra <= RX1_douta(38 DOWNTO 32);
            SP2a_wea <= '0';
            SP2b_wea <= '0';
            SP2a_addra <= RX2_douta(46 DOWNTO 40);
            SP2b_addra <= RX2_douta(38 DOWNTO 32);
            SP3a_wea <= '0';
            SP3b_wea <= '0';
            SP3a_addra <= RX3_douta(46 DOWNTO 40);
            SP3b_addra <= RX3_douta(38 DOWNTO 32);
            SP4a_wea <= '0';
            SP4b_wea <= '0';
            SP4a_addra <= RX4_douta(46 DOWNTO 40);
            SP4b_addra <= RX4_douta(38 DOWNTO 32);
            SP5a_wea <= '0';
            SP5b_wea <= '0';
            SP5a_addra <= RX5_douta(46 DOWNTO 40);
            SP5b_addra <= RX5_douta(38 DOWNTO 32);
            SP6a_wea <= '0';
            SP6b_wea <= '0';
            SP6a_addra <= RX6_douta(46 DOWNTO 40);
            SP6b_addra <= RX6_douta(38 DOWNTO 32);
            SP7a_wea <= '0';
            SP7b_wea <= '0';
            SP7a_addra <= RX7_douta(46 DOWNTO 40);
```

```
            SP7b_addra <= RX7_douta(38 DOWNTO 32);
            SP8a_wea <= '0';
            SP8b_wea <= '0';
            SP8a_addra <= RX8_douta(46 DOWNTO 40);
            SP8b_addra <= RX8_douta(38 DOWNTO 32);

            v_PC1_POP1 := SP1a_douta;
            v_PC1_POP2 := SP1b_douta;
            v_PC1_RX := RX1_douta;
            v_PC2_POP1 := SP2a_douta;
            v_PC2_POP2 := SP2b_douta;
            v_PC2_RX := RX2_douta;
            v_PC3_POP1 := SP3a_douta;
            v_PC3_POP2 := SP3b_douta;
            v_PC3_RX := RX3_douta;
            v_PC4_POP1 := SP4a_douta;
            v_PC4_POP2 := SP4b_douta;
            v_PC4_RX := RX4_douta;
            v_PC5_POP1 := SP5a_douta;
            v_PC5_POP2 := SP5b_douta;
            v_PC5_RX := RX5_douta;
            v_PC6_POP1 := SP6a_douta;
            v_PC6_POP2 := SP6b_douta;
            v_PC6_RX := RX6_douta;
            v_PC7_POP1 := SP7a_douta;
            v_PC7_POP2 := SP7b_douta;
            v_PC7_RX := RX7_douta;
            v_PC8_POP1 := SP8a_douta;
            v_PC8_POP2 := SP8b_douta;
            v_PC8_RX := RX8_douta;
            state2 := state2 + 1;
        ELSIF (state2 = "00000110") THEN
            state2 := "00000000";
            P1_wea <= '1';
            P1_addra <= LBOUND_1 + count;
            P1_dina <= PC1_PROP;
            P2_wea <= '1';
            P2_addra <= LBOUND_2 + count;
            P2_dina <= PC2_PROP;
            P3_wea <= '1';
            P3_addra <= LBOUND_3 + count;
            P3_dina <= PC3_PROP;
            P4_wea <= '1';
            P4_addra <= LBOUND_4 + count;
            P4_dina <= PC4_PROP;
            P5_wea <= '1';
            P5_addra <= LBOUND_5 + count;
            P5_dina <= PC5_PROP;
            P6_wea <= '1';
            P6_addra <= LBOUND_6 + count;
            P6_dina <= PC6_PROP;
            P7_wea <= '1';
            P7_addra <= LBOUND_7 + count;
            P7_dina <= PC7_PROP;
            P8_wea <= '1';
            P8_addra <= LBOUND_8 + count;
            P8_dina <= PC8_PROP;
```

```
                IF (count = "000000") THEN
                      v_PSUM1_1 := PC1_PROP;
                      v_PSUM2_1 := PC2_PROP;
                      v_PSUM3_1 := PC3_PROP;
                      v_PSUM4_1 := PC4_PROP;
                      v_PSUM5_1 := PC5_PROP;
                      v_PSUM6_1 := PC6_PROP;
                      v_PSUM7_1 := PC7_PROP;
                      v_PSUM8_1 := PC8_PROP;
                      count := count + 1;
                ELSIF (count < "000100") THEN
                      v_PSUM1_1 := v_PSUM1_1 + PC1_PROP;
                      v_PSUM2_1 := v_PSUM2_1 + PC2_PROP;
                      v_PSUM3_1 := v_PSUM3_1 + PC3_PROP;
                      v_PSUM4_1 := v_PSUM4_1 + PC4_PROP;
                      v_PSUM5_1 := v_PSUM5_1 + PC5_PROP;
                      v_PSUM6_1 := v_PSUM6_1 + PC6_PROP;
                      v_PSUM7_1 := v_PSUM7_1 + PC7_PROP;
                      v_PSUM8_1 := v_PSUM8_1 + PC8_PROP;
                      count := count + 1;
                ELSIF (count = "000100") THEN
                      v_PSUM1_2 := v_PSUM1_1 + PC1_PROP;
                      v_PSUM2_2 := v_PSUM2_1 + PC2_PROP;
                      v_PSUM3_2 := v_PSUM3_1 + PC3_PROP;
                      v_PSUM4_2 := v_PSUM4_1 + PC4_PROP;
                      v_PSUM5_2 := v_PSUM5_1 + PC5_PROP;
                      v_PSUM6_2 := v_PSUM6_1 + PC6_PROP;
                      v_PSUM7_2 := v_PSUM7_1 + PC7_PROP;
                      v_PSUM8_2 := v_PSUM8_1 + PC8_PROP;
                      count := count + 1;
                ELSIF (count < "000111") THEN
                      v_PSUM1_2 := v_PSUM1_2 + PC1_PROP;
                      v_PSUM2_2 := v_PSUM2_2 + PC2_PROP;
                      v_PSUM3_2 := v_PSUM3_2 + PC3_PROP;
                      v_PSUM4_2 := v_PSUM4_2 + PC4_PROP;
                      v_PSUM5_2 := v_PSUM5_2 + PC5_PROP;
                      v_PSUM6_2 := v_PSUM6_2 + PC6_PROP;
                      v_PSUM7_2 := v_PSUM7_2 + PC7_PROP;
                      v_PSUM8_2 := v_PSUM8_2 + PC8_PROP;
                      count := count + 1;
                ELSE
                      v_PSUM1_2 := v_PSUM1_2 + PC1_PROP;
                      v_PSUM2_2 := v_PSUM2_2 + PC2_PROP;
                      v_PSUM3_2 := v_PSUM3_2 + PC3_PROP;
                      v_PSUM4_2 := v_PSUM4_2 + PC4_PROP;
                      v_PSUM5_2 := v_PSUM5_2 + PC5_PROP;
                      v_PSUM6_2 := v_PSUM6_2 + PC6_PROP;
                      v_PSUM7_2 := v_PSUM7_2 + PC7_PROP;
                      v_PSUM8_2 := v_PSUM8_2 + PC8_PROP;
                      state := state + 1;
                      count := "000000";
                END IF;
          ELSE
                state2 := state2 + 1;
          END IF;

    -- STEP THROUGH 125 REACTIONS
```

```
            ELSIF (dout(63 DOWNTO 59) = "01110") THEN
--                  we <= '0';
--                  addr <= X"00";
                    index := X"02";
                    maxindex := dout(7 DOWNTO 0);
                    looping := '1';
                    we <= '1';
                    addr <= X"01";
                    din(63 DOWNTO 40) <= X"000000";
                    din(39 DOWNTO 32) <= index;
                    din(31 DOWNTO 8) <= X"000000";
                    din(7 DOWNTO 0) <= maxindex;
                    state2 := "00000000";

            END IF;

      -- TELL CPU THAT FPGA IS DONE
      ELSIF (state = "0010") THEN
            we <= '1';
            addr <= X"00";
            din <= (others => '0');
            SPUS1_wea <= '0';
            SPUS2_wea <= '0';
            SPUS3_wea <= '0';
            SPUS4_wea <= '0';
            SP1a_wea <= '0';
            SP1b_wea <= '0';
            SP2a_wea <= '0';
            SP2b_wea <= '0';
            SP3a_wea <= '0';
            SP3b_wea <= '0';
            SP4a_wea <= '0';
            SP4b_wea <= '0';
            SP5a_wea <= '0';
            SP5b_wea <= '0';
            SP6a_wea <= '0';
            SP6b_wea <= '0';
            SP7a_wea <= '0';
            SP7b_wea <= '0';
            SP8a_wea <= '0';
            SP8b_wea <= '0';
            RX1_wea <= '0';
            RX2_wea <= '0';
            RX3_wea <= '0';
            RX4_wea <= '0';
            RX5_wea <= '0';
            RX6_wea <= '0';
            RX7_wea <= '0';
            RX8_wea <= '0';
            RXUS_wea <= '0';
            P1_wea <= '0';
            P2_wea <= '0';
            P3_wea <= '0';
            P4_wea <= '0';
            P5_wea <= '0';
            P6_wea <= '0';
            P7_wea <= '0';
```

```
                P8_wea <= '0';
                state := "00000000";
            ELSE
                we <= '0';
                addr <= X"00";
                state := state + 1;
            END IF;

            product <= theproduct;
            PSUM1_1 <= v_PSUM1_1; PSUM1_2 <= v_PSUM1_2; PSUM2_1 <=
v_PSUM2_1; PSUM2_2 <= v_PSUM2_2;
            PSUM3_1 <= v_PSUM3_1; PSUM3_2 <= v_PSUM3_2; PSUM4_1 <=
v_PSUM4_1; PSUM4_2 <= v_PSUM4_2;
            PSUM5_1 <= v_PSUM5_1; PSUM5_2 <= v_PSUM5_2; PSUM6_1 <=
v_PSUM6_1; PSUM6_2 <= v_PSUM6_2;
            PSUM7_1 <= v_PSUM7_1; PSUM7_2 <= v_PSUM7_2; PSUM8_1 <=
v_PSUM8_1; PSUM8_2 <= v_PSUM8_2;

            PC1_POP1 <= v_PC1_POP1; PC1_POP2 <= v_PC1_POP2; PC1_RX <=
v_PC1_RX;
            PC2_POP1 <= v_PC2_POP1; PC2_POP2 <= v_PC2_POP2; PC2_RX <=
v_PC2_RX;
            PC3_POP1 <= v_PC3_POP1; PC3_POP2 <= v_PC3_POP2; PC3_RX <=
v_PC3_RX;
            PC4_POP1 <= v_PC4_POP1; PC4_POP2 <= v_PC4_POP2; PC4_RX <=
v_PC4_RX;
            PC5_POP1 <= v_PC5_POP1; PC5_POP2 <= v_PC5_POP2; PC5_RX <=
v_PC5_RX;
            PC6_POP1 <= v_PC6_POP1; PC6_POP2 <= v_PC6_POP2; PC6_RX <=
v_PC6_RX;
            PC7_POP1 <= v_PC7_POP1; PC7_POP2 <= v_PC7_POP2; PC7_RX <=
v_PC7_RX;
            PC8_POP1 <= v_PC8_POP1; PC8_POP2 <= v_PC8_POP2; PC8_RX <=
v_PC8_RX;


        END IF;
    END PROCESS;


-- PROCESS TO DETERMINE THE ADDRESS OF THE NEXT REACTION
    PROCESS (clk)
        VARIABLE nextreac : STD_LOGIC_VECTOR(5 DOWNTO 0);
        VARIABLE v_rxselect : STD_LOGIC_VECTOR(5 DOWNTO 0);
        VARIABLE baddr : STD_LOGIC_VECTOR(5 DOWNTO 0);
        VARIABLE count : STD_LOGIC_VECTOR(5 DOWNTO 0);
        VARIABLE apro : STD_LOGIC_VECTOR(47 DOWNTO 0);
        VARIABLE rx_state : STD_LOGIC_VECTOR(3 DOWNTO 0);
        VARIABLE pcolumn : STD_LOGIC_VECTOR(3 DOWNTO 0);

    BEGIN
        IF (clk'EVENT AND clk='1') THEN
            P1_web <= '0'; P2_web <= '0'; P3_web <= '0'; P4_web <= '0';
            P5_web <= '0'; P6_web <= '0'; P7_web <= '0'; P8_web <= '0';

            IF (product(79 DOWNTO 32) < PSUM1_2) THEN
                --IF (pcolumn /= X"1") THEN
```

```
        --    pcolumn := X"1";
        --    rx_state := X"0";
        --END IF;
        CASE rx_state IS
        WHEN "0000" =>
              apro := product(79 DOWNTO 32);
              rx_state := rx_state + 1;
              count := "000000";
        WHEN "0001" =>
              IF (apro < PSUM1_1) THEN
                    baddr := "000000";
              ELSE
                    baddr := "000100";
                    apro := apro - PSUM1_1;
              END IF;
              P1_addrb <= baddr;
              rx_state := rx_state + 1;
        WHEN "0010" =>
              P1_addrb <= baddr + count;
              rx_state := rx_state + 1;
        WHEN "0011" =>
              IF (apro < P1_doutb) THEN
                    nextreac := baddr + count;
                    rx_state := rx_state + 1;
              ELSIF (count = "000011") THEN
                    nextreac := baddr + count;
                    rx_state := rx_state + 1;
              ELSE
                    apro := apro - P1_doutb;
                    count := count + 1;
                    P1_addrb <= baddr + count;
                    rx_state := X"2";
              END IF;
        WHEN "0100" =>
                    v_rxselect := nextreac;
                    rx_state := X"0";
        WHEN OTHERS =>
              P1_addrb <= baddr + count;
              rx_state := X"0";
        END CASE;

  ELSIF (product(79 DOWNTO 32) < TPROP2) THEN
        --IF (pcolumn /= X"2") THEN
        --    pcolumn := X"2";
        --    rx_state := X"0";
        --END IF;
        CASE rx_state IS
        WHEN "0000" =>
              apro := product(79 DOWNTO 32) - PSUM1_2;
              rx_state := rx_state + 1;
              count := "000000";
        WHEN "0001" =>
              IF (apro < PSUM2_1) THEN
                    baddr := "001000";
              ELSE
                    apro := apro - PSUM2_1;
                    baddr := "001100";
```

```
                END IF;
                P2_addrb <= baddr;
                rx_state := rx_state + 1;
          WHEN "0010" =>
                P2_addrb <= baddr + count;
                rx_state := rx_state + 1;
          WHEN "0011" =>
                IF (apro < P2_doutb) THEN
                      nextreac := baddr + count;
                      rx_state := rx_state + 1;
                ELSIF (count = "000011") THEN
                      nextreac := baddr + count;
                      rx_state := rx_state + 1;
                ELSE
                      apro := apro - P2_doutb;
                      count := count + 1;
                      P2_addrb <= baddr + count;
                      rx_state := X"2";
                END IF;
          WHEN "0100" =>
                      v_rxselect := nextreac;
                      rx_state := X"0";
          WHEN OTHERS =>
                P2_addrb <= baddr + count;
                rx_state := X"0";
          END CASE;

    ELSIF (product(79 DOWNTO 32) < TPROP3) THEN
          --IF (pcolumn /= X"3") THEN
          --    pcolumn := X"3";
          --    rx_state := X"0";
          --END IF;
          CASE rx_state IS
          WHEN "0000" =>
                apro := product(79 DOWNTO 32) - TPROP2;
                rx_state := rx_state + 1;
                count := "000000";
          WHEN "0001" =>
                IF (apro < PSUM3_1) THEN
                      baddr := "010000";
                ELSE
                      apro := apro - PSUM3_1;
                      baddr := "010100";
                END IF;
                P3_addrb <= baddr;
                rx_state := rx_state + 1;
          WHEN "0010" =>
                P3_addrb <= baddr + count;
                rx_state := rx_state + 1;
          WHEN "0011" =>
                IF (apro < P3_doutb) THEN
                      nextreac := baddr + count;
                      rx_state := rx_state + 1;
                ELSIF (count = "000011") THEN
                      nextreac := baddr + count;
                      rx_state := rx_state + 1;
                ELSE
```

```
                  apro := apro - P3_doutb;
                  count := count + 1;
                  P3_addrb <= baddr + count;
                  rx_state := X"2";
            END IF;
      WHEN "0100" =>
                  v_rxselect := nextreac;
                  rx_state := X"0";
      WHEN OTHERS =>
            P3_addrb <= baddr + count;
            rx_state := X"0";
      END CASE;

ELSIF (product(79 DOWNTO 32) < TPROP4) THEN
      --IF (pcolumn /= X"4") THEN
      --    pcolumn := X"4";
      --    rx_state := X"0";
      --END IF;
      CASE rx_state IS
      WHEN "0000" =>
            apro := product(79 DOWNTO 32) - TPROP3;
            rx_state := rx_state + 1;
            count := "000000";
      WHEN "0001" =>
            IF (apro < PSUM4_1) THEN
                  baddr := "011000";
            ELSE
                  apro := apro - PSUM4_1;
                  baddr := "011100";
            END IF;
            P4_addrb <= baddr;
            rx_state := rx_state + 1;
      WHEN "0010" =>
            P4_addrb <= baddr + count;
            rx_state := rx_state + 1;
      WHEN "0011" =>
            IF (apro < P4_doutb) THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
            ELSIF (count = "000011") THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
            ELSE
                  apro := apro - P4_doutb;
                  count := count + 1;
                  P4_addrb <= baddr + count;
                  rx_state := X"2";
            END IF;
      WHEN "0100" =>
                  v_rxselect := nextreac;
                  rx_state := X"0";
      WHEN OTHERS =>
            P4_addrb <= baddr + count;
            rx_state := X"0";
      END CASE;

ELSIF (product(79 DOWNTO 32) < TPROP5) THEN
```

```
            --IF (pcolumn /= X"5") THEN
            --    pcolumn := X"5";
            --    rx_state := X"0";
            --END IF;
            CASE rx_state IS
            WHEN "0000" =>
                  apro := product(79 DOWNTO 32) - TPROP4;
                  rx_state := rx_state + 1;
                  count := "000000";
            WHEN "0001" =>
                  IF (apro < PSUM5_1) THEN
                        baddr := "100000";
                  ELSE
                        apro := apro - PSUM5_1;
                        baddr := "100100";
                  END IF;
                  P5_addrb <= baddr;
                  rx_state := rx_state + 1;
            WHEN "0010" =>
                  P5_addrb <= baddr + count;
                  rx_state := rx_state + 1;
            WHEN "0011" =>
                  IF (apro < P5_doutb) THEN
                        nextreac := baddr + count;
                        rx_state := rx_state + 1;
                  ELSIF (count = "000011") THEN
                        nextreac := baddr + count;
                        rx_state := rx_state + 1;
                  ELSE
                        apro := apro - P5_doutb;
                        count := count + 1;
                        P5_addrb <= baddr + count;
                        rx_state := X"2";
                  END IF;
            WHEN "0100" =>
                        v_rxselect := nextreac;
                        rx_state := X"0";
            WHEN OTHERS =>
                  P5_addrb <= baddr + count;
                  rx_state := X"0";
            END CASE;

      ELSIF (product(79 DOWNTO 32) < TPROP6) THEN
            --IF (pcolumn /= X"6") THEN
            --    pcolumn := X"6";
            --    rx_state := X"0";
            --END IF;
            CASE rx_state IS
            WHEN "0000" =>
                  apro := product(79 DOWNTO 32) - TPROP5;
                  rx_state := rx_state + 1;
                  count := "000000";
            WHEN "0001" =>
                  IF (apro < PSUM6_1) THEN
                        baddr := "101000";
                  ELSE
                        apro := apro - PSUM6_1;
```

```
                  baddr := "101100";
            END IF;
            P6_addrb <= baddr;
            rx_state := rx_state + 1;
      WHEN "0010" =>
            P6_addrb <= baddr + count;
            rx_state := rx_state + 1;
      WHEN "0011" =>
            IF (apro < P6_doutb) THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
            ELSIF (count = "000011") THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
            ELSE
                  apro := apro - P6_doutb;
                  count := count + 1;
                  P6_addrb <= baddr + count;
                  rx_state := X"2";
            END IF;
      WHEN "0100" =>
                  v_rxselect := nextreac;
                  rx_state := X"0";
      WHEN OTHERS =>
            P6_addrb <= baddr + count;
            rx_state := X"0";
      END CASE;

ELSIF (product(79 DOWNTO 32) < TPROP7) THEN
      --IF (pcolumn /= X"7") THEN
      --    pcolumn := X"7";
      --    rx_state := X"0";
      --END IF;
      CASE rx_state IS
      WHEN "0000" =>
            apro := product(79 DOWNTO 32) - TPROP6;
            rx_state := rx_state + 1;
            count := "000000";
      WHEN "0001" =>
            IF (apro < PSUM7_1) THEN
                  baddr := "110000";
            ELSE
                  apro := apro - PSUM7_1;
                  baddr := "110100";
            END IF;
            P7_addrb <= baddr;
            rx_state := rx_state + 1;
      WHEN "0010" =>
            P7_addrb <= baddr + count;
            rx_state := rx_state + 1;
      WHEN "0011" =>
            IF (apro < P7_doutb) THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
            ELSIF (count = "000011") THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
```

```
            ELSE
                  apro := apro - P7_doutb;
                  count := count + 1;
                  P7_addrb <= baddr + count;
                  rx_state := X"2";
            END IF;
      WHEN "0100" =>
                  v_rxselect := nextreac;
                  rx_state := X"0";
      WHEN OTHERS =>
            P7_addrb <= baddr + count;
            rx_state := X"0";
      END CASE;

ELSE
      --IF (pcolumn /= X"8") THEN
      --    pcolumn := X"8";
      --    rx_state := X"0";
      --END IF;
      CASE rx_state IS
      WHEN "0000" =>
            apro := product(79 DOWNTO 32) - TPROP7;
            rx_state := rx_state + 1;
            count := "000000";
      WHEN "0001" =>
            IF (apro < PSUM8_1) THEN
                  baddr := "111000";
            ELSE
                  apro := apro - PSUM8_1;
                  baddr := "111100";
            END IF;
            P8_addrb <= baddr;
            rx_state := rx_state + 1;
      WHEN "0010" =>
            P8_addrb <= baddr + count;
            rx_state := rx_state + 1;
      WHEN "0011" =>
            IF (apro < P8_doutb) THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
            ELSIF (count = "000011") THEN
                  nextreac := baddr + count;
                  rx_state := rx_state + 1;
            ELSE
                  apro := apro - P8_doutb;
                  count := count + 1;
                  P8_addrb <= baddr + count;
                  rx_state := X"2";
            END IF;
      WHEN "0100" =>
                  v_rxselect := nextreac;
                  rx_state := X"0";
      WHEN OTHERS =>
            P8_addrb <= baddr + count;
            rx_state := X"0";
      END CASE;
```

```
            END IF;

            s_rxselect <= v_rxselect;
        END IF;
    END PROCESS;

END rtl;
```

propcalc.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY propcalc IS
  PORT (
      clk               : IN STD_LOGIC;
      POP1        : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      POP2        : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      RX                : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
      PROPENSITY  : OUT STD_LOGIC_VECTOR(47 DOWNTO 0) );
END propcalc;

ARCHITECTURE rtl OF propcalc IS

BEGIN

  PROCESS(clk)
      VARIABLE X,Y      : STD_LOGIC_VECTOR(15 DOWNTO 0);
      VARIABLE prop     : STD_LOGIC_VECTOR(47 DOWNTO 0);

  BEGIN
      IF (clk'EVENT AND clk='0') THEN
                  X := POP1;
                  IF (RX(47 DOWNTO 40) = RX(39 DOWNTO 32)) THEN
                              Y := X - 1;
                  ELSE
                        Y := POP2;
                  END IF;
                  prop := RX(15 DOWNTO 0) * X * Y;
                  IF (RX(47 DOWNTO 40) = RX(39 DOWNTO 32)) THEN
                        prop(46 DOWNTO 0) := prop(47 DOWNTO 1);
                        prop(47) := '0';
                  END IF;
                  PROPENSITY <= prop;
      END IF;
  END PROCESS;
END rtl;
```

sumprop.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

ENTITY sumprop IS
  PORT (    clk   : IN STD_LOGIC;
            PSUM1 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM2 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM3 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM4 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM5 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM6 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM7 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            PSUM8 : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL2      : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL3      : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL4      : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL5      : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL6      : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL7      : OUT STD_LOGIC_VECTOR(47 DOWNTO 0);
            TOTAL8      : OUT STD_LOGIC_VECTOR(47 DOWNTO 0) );
END sumprop;

ARCHITECTURE rtl OF sumprop IS

BEGIN
  PROCESS(clk)
      VARIABLE sum2 : STD_LOGIC_VECTOR(47 DOWNTO 0);
      VARIABLE sum3 : STD_LOGIC_VECTOR(47 DOWNTO 0);
      VARIABLE sum4 : STD_LOGIC_VECTOR(47 DOWNTO 0);
      VARIABLE sum5 : STD_LOGIC_VECTOR(47 DOWNTO 0);
      VARIABLE sum6 : STD_LOGIC_VECTOR(47 DOWNTO 0);
      VARIABLE sum7 : STD_LOGIC_VECTOR(47 DOWNTO 0);
      VARIABLE sum8 : STD_LOGIC_VECTOR(47 DOWNTO 0);
  BEGIN
      IF (clk'EVENT AND clk='1') THEN
            sum2 := PSUM1 + PSUM2;
            sum3 := PSUM1 + PSUM2 + PSUM3;
            sum4 := PSUM1 + PSUM2 + PSUM3 + PSUM4;
            sum5 := PSUM1 + PSUM2 + PSUM3 + PSUM4 + PSUM5;
            sum6 := PSUM1 + PSUM2 + PSUM3 + PSUM4 + PSUM5 + PSUM6;
            sum7 := PSUM1 + PSUM2 + PSUM3 + PSUM4 + PSUM5 + PSUM6 +
PSUM7;
            sum8 := PSUM1 + PSUM2 + PSUM3 + PSUM4 + PSUM5 + PSUM6 +
PSUM7 + PSUM8;

            TOTAL2 <= sum2; TOTAL3 <= sum3; TOTAL4 <= sum4;
            TOTAL5 <= sum5; TOTAL6 <= sum6; TOTAL7 <= sum7;
            TOTAL8 <= sum8;

      END IF;
  END PROCESS;
END rtl;
```

lfsr32.vhd [17]

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity lfsr32 is
  port   ( in_clock          : in std_logic;
         in_reset          : in std_logic;
         in_seed           : in std_logic_vector(31 downto 0);
         out_random_number : out std_logic_vector(31 downto 0));
end entity lfsr32;

architecture a of lfsr32 is
begin
  process(in_clock)
    variable var_current_number : std_logic_vector(31 downto 0);
    variable var_startup : natural;
    variable var_next_bit : std_logic;
  begin
    if (in_clock = '1' and in_clock'event) then
      if (in_reset='1' or var_startup=0) then
      var_current_number := in_seed;
      var_startup := 1;
      else
      var_next_bit := var_current_number(0) XOR
                  var_current_number(26) XOR
                  var_current_number(27) XOR
                  var_current_number(31);
      var_current_number(31 downto 1) := var_current_number(30 downto
0);
      var_current_number(0) := var_next_bit;
      end if;
      out_random_number <= var_current_number;
    end if;
  end process;
end architecture a;
```

exp_rand.vhd [17]

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity exp_rand is
  port ( in_clock : in std_logic;
       out_uniform_number : out std_logic_vector(31 downto 0);
       out_random_number : out std_logic_vector(31 downto 0));
end entity;

architecture a of exp_rand is
  component linear_interp
    port (in_clk : in std_logic;
        in_rand : in std_logic_vector(15 downto 0);
        in_min : in std_logic_vector(15 downto 0);
        in_diff : in std_logic_vector(15 downto 0);
        in_urn : in std_logic_vector(31 downto 0);
        out_urn : out std_logic_vector(31 downto 0);
        out_interp : out std_logic_vector(31 downto 0));
  end component;

  component lfsr32
    port   ( in_clock          : in std_logic;
           in_reset          : in std_logic;
           in_seed           : in std_logic_vector(31 downto 0);
           out_random_number : out std_logic_vector(31 downto 0));
  end component;

  component negative_log_lut
    port(index : in std_logic_vector(7 downto 0);
      in_urn : in std_logic_vector(31 downto 0);
      out_urn : out std_logic_vector(31 downto 0);
       min : out std_logic_vector(15 downto 0);
       diff : out std_logic_vector(15 downto 0));
  end component;

  signal sig_0 : std_logic;
  signal sig_expseed : std_logic_vector(31 downto 0);
  signal sig_urn : std_logic_vector(31 downto 0);
  signal sig_outurn : std_logic_vector(31 downto 0);
  signal sig_min : std_logic_vector(15 downto 0);
  signal sig_diff : std_logic_vector(15 downto 0);

begin
  sig_0 <= '0';
  sig_expseed <= "10101010101010101010101010101010";

  m0 : lfsr32
    port map(in_clock,sig_0,sig_expseed,sig_urn);

  m1 : linear_interp
    port map(in_clock,sig_urn(15 downto 0),sig_min,
sig_diff,sig_outurn,out_uniform_number,out_random_number);
```

```
  m2 : negative_log_lut
    port map(sig_urn(23 downto
16),sig_urn,sig_outurn,sig_min,sig_diff);

end;
```

negative_log_lut.vhd [17]

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity negative_log_lut is
  port(index : in std_logic_vector(7 downto 0);
            in_urn : in std_logic_vector(31 downto 0);
            out_urn : out std_logic_vector(31 downto 0);
       min : out std_logic_vector(15 downto 0);
       diff : out std_logic_vector(15 downto 0));
end entity;

architecture a of negative_log_lut is
begin
  process(index)
  begin
      out_urn <= in_urn;
    case index is
      when "00000000" =>
        diff <= "0100111010001101";
        min <=  "1011000101110010";
      when "00000001" =>
        diff <= "0001011000101110";
        min <= "1001101101000011";
      when "00000010" =>
        diff <= "0000110011111001";
        min <= "1000111001001010";
      when "00000011" =>
        diff <= "0000100100110100";
        min <= "1000010100010101";
      when "00000100" =>
        diff <= "0000011100100011";
        min <= "0111110111110001";
      when "00000101" =>
        diff <= "0000010111010101";
        min <= "0111100000011100";
      when "00000110" =>
        diff <= "0000010011101110";
        min <= "0111001100101101";
      when "00000111" =>
        diff <= "0000010001000101";
        min <= "0110111011100111";
      when "00001000" =>
        diff <= "0000001111000100";
        min <= "0110101100100010";
      when "00001001" =>
        diff <= "0000001101011111";
        min <= "0110011111000011";
      when "00001010" =>
        diff <= "0000001100001100";
        min <= "0110010010110110";
      when "00001011" =>
        diff <= "0000001011001000";
```

```
  min <= "0110000111101101";
when "00001100" =>
  diff <= "0000001010001111";
  min <= "0101111101011110";
when "00001101" =>
  diff <= "0000001001011111";
  min <= "0101110011111110";
when "00001110" =>
  diff <= "0000001000110101";
  min <= "0101101011001001";
when "00001111" =>
  diff <= "0000001000010000";
  min <= "0101100010111001";
when "00010000" =>
  diff <= "0000000111110000";
  min <= "0101011011001000";
when "00010001" =>
  diff <= "0000000111010100";
  min <= "0101010011110100";
when "00010010" =>
  diff <= "0000000110111010";
  min <= "0101001100111001";
when "00010011" =>
  diff <= "0000000110100100";
  min <= "0101000110010101";
when "00010100" =>
  diff <= "0000000110001111";
  min <= "0101000000000101";
when "00010101" =>
  diff <= "0000000101111101";
  min <= "0100111010001000";
when "00010110" =>
  diff <= "0000000101101100";
  min <= "0100110100011100";
when "00010111" =>
  diff <= "0000000101011100";
  min <= "0100101110111111";
when "00011000" =>
  diff <= "0000000101001110";
  min <= "0100101001110001";
when "00011001" =>
  diff <= "0000000101000001";
  min <= "0100100100101111";
when "00011010" =>
  diff <= "0000000100110101";
  min <= "0100011111111010";
when "00011011" =>
  diff <= "0000000100101001";
  min <= "0100011011010000";
when "00011100" =>
  diff <= "0000000100011111";
  min <= "0100010110110001";
when "00011101" =>
  diff <= "0000000100010101";
  min <= "0100010010011011";
when "00011110" =>
  diff <= "0000000100001100";
```

```
  min <= "0100001110001110";
when "00011111" =>
  diff <= "0000000100000100";
  min <= "0100001010001010";
when "00100000" =>
  diff <= "0000000011111100";
  min <= "0100000110001110";
when "00100001" =>
  diff <= "0000000011110100";
  min <= "0100000010011010";
when "00100010" =>
  diff <= "0000000011101101";
  min <= "0011111110101100";
when "00100011" =>
  diff <= "0000000011100110";
  min <= "0011111011000101";
when "00100100" =>
  diff <= "0000000011100000";
  min <= "0011110111100101";
when "00100101" =>
  diff <= "0000000011011010";
  min <= "0011110100001010";
when "00100110" =>
  diff <= "0000000011010100";
  min <= "0011110000110110";
when "00100111" =>
  diff <= "0000000011001111";
  min <= "0011101101100110";
when "00101000" =>
  diff <= "0000000011001010";
  min <= "0011101010011100";
when "00101001" =>
  diff <= "0000000011000101";
  min <= "0011100111010111";
when "00101010" =>
  diff <= "0000000011000000";
  min <= "0011100100010110";
when "00101011" =>
  diff <= "0000000010111100";
  min <= "0011100001011010";
when "00101100" =>
  diff <= "0000000010111000";
  min <= "0011011110100001";
when "00101101" =>
  diff <= "0000000010110100";
  min <= "0011011011101101";
when "00101110" =>
  diff <= "0000000010110000";
  min <= "0011011000111101";
when "00101111" =>
  diff <= "0000000010101100";
  min <= "0011010110010001";
when "00110000" =>
  diff <= "0000000010101000";
  min <= "0011010011101000";
when "00110001" =>
  diff <= "0000000010100101";
```

```
  min <= "0011010001000010";
when "00110010" =>
  diff <= "0000000010100010";
  min <= "0011001110100000";
when "00110011" =>
  diff <= "0000000010011111";
  min <= "0011001100000001";
when "00110100" =>
  diff <= "0000000010011100";
  min <= "0011001001100101";
when "00110101" =>
  diff <= "0000000010011001";
  min <= "0011000111001100";
when "00110110" =>
  diff <= "0000000010010110";
  min <= "0011000100110110";
when "00110111" =>
  diff <= "0000000010010011";
  min <= "0011000010100010";
when "00111000" =>
  diff <= "0000000010010000";
  min <= "0011000000010001";
when "00111001" =>
  diff <= "0000000010001110";
  min <= "0010111110000010";
when "00111010" =>
  diff <= "0000000010001100";
  min <= "0010111011110110";
when "00111011" =>
  diff <= "0000000010001001";
  min <= "0010111001101101";
when "00111100" =>
  diff <= "0000000010000111";
  min <= "0010110111100101";
when "00111101" =>
  diff <= "0000000010000101";
  min <= "0010110101100000";
when "00111110" =>
  diff <= "0000000010000011";
  min <= "0010110011011101";
when "00111111" =>
  diff <= "0000000010000001";
  min <= "0010110001011100";
when "01000000" =>
  diff <= "0000000001111111";
  min <= "0010101111011101";
when "01000001" =>
  diff <= "0000000001111101";
  min <= "0010101101100000";
when "01000010" =>
  diff <= "0000000001111011";
  min <= "0010101011100101";
when "01000011" =>
  diff <= "0000000001111001";
  min <= "0010101001101011";
when "01000100" =>
  diff <= "0000000001110111";
```

```
  min <= "0010100111110100";
when "01000101" =>
  diff <= "0000000001110101";
  min <= "0010100101111110";
when "01000110" =>
  diff <= "0000000001110100";
  min <= "0010100100001010";
when "01000111" =>
  diff <= "0000000001110010";
  min <= "0010100010010111";
when "01001000" =>
  diff <= "0000000001110000";
  min <= "0010100000100110";
when "01001001" =>
  diff <= "0000000001101111";
  min <= "0010011110110111";
when "01001010" =>
  diff <= "0000000001101101";
  min <= "0010011101001001";
when "01001011" =>
  diff <= "0000000001101100";
  min <= "0010011011011100";
when "01001100" =>
  diff <= "0000000001101011";
  min <= "0010011001110001";
when "01001101" =>
  diff <= "0000000001101001";
  min <= "0010011000000111";
when "01001110" =>
  diff <= "0000000001101000";
  min <= "0010010110011111";
when "01001111" =>
  diff <= "0000000001100111";
  min <= "0010010100111000";
when "01010000" =>
  diff <= "0000000001100101";
  min <= "0010010011010010";
when "01010001" =>
  diff <= "0000000001100100";
  min <= "0010010001101110";
when "01010010" =>
  diff <= "0000000001100011";
  min <= "0010010000001010";
when "01010011" =>
  diff <= "0000000001100010";
  min <= "0010001110101000";
when "01010100" =>
  diff <= "0000000001100000";
  min <= "0010001101000111";
when "01010101" =>
  diff <= "0000000001011111";
  min <= "0010001011101000";
when "01010110" =>
  diff <= "0000000001011110";
  min <= "0010001010001001";
when "01010111" =>
  diff <= "0000000001011101";
```

```
  min <= "0010001000101011";
when "01011000" =>
  diff <= "0000000001011100";
  min <= "0010000111001111";
when "01011001" =>
  diff <= "0000000001011011";
  min <= "0010000101110011";
when "01011010" =>
  diff <= "0000000001011010";
  min <= "0010000100011001";
when "01011011" =>
  diff <= "0000000001011001";
  min <= "0010000010111111";
when "01011100" =>
  diff <= "0000000001011000";
  min <= "0010000001100111";
when "01011101" =>
  diff <= "0000000001010111";
  min <= "0010000000001111";
when "01011110" =>
  diff <= "0000000001010110";
  min <= "0001111110111000";
when "01011111" =>
  diff <= "0000000001010101";
  min <= "0001111101100010";
when "01100000" =>
  diff <= "0000000001010100";
  min <= "0001111100001110";
when "01100001" =>
  diff <= "0000000001010100";
  min <= "0001111010111010";
when "01100010" =>
  diff <= "0000000001010011";
  min <= "0001111001100110";
when "01100011" =>
  diff <= "0000000001010010";
  min <= "0001111000010100";
when "01100100" =>
  diff <= "0000000001010001";
  min <= "0001110111000011";
when "01100101" =>
  diff <= "0000000001010000";
  min <= "0001110101110010";
when "01100110" =>
  diff <= "0000000001001111";
  min <= "0001110100100010";
when "01100111" =>
  diff <= "0000000001001111";
  min <= "0001110011010011";
when "01101000" =>
  diff <= "0000000001001110";
  min <= "0001110010000100";
when "01101001" =>
  diff <= "0000000001001101";
  min <= "0001110000110111";
when "01101010" =>
  diff <= "0000000001001100";
```

```
  min <= "0001101111101010";
when "01101011" =>
  diff <= "0000000001001100";
  min <= "0001101110011110";
when "01101100" =>
  diff <= "0000000001001011";
  min <= "0001101101010010";
when "01101101" =>
  diff <= "0000000001001010";
  min <= "0001101100000111";
when "01101110" =>
  diff <= "0000000001001010";
  min <= "0001101010111101";
when "01101111" =>
  diff <= "0000000001001001";
  min <= "0001101001110100";
when "01110000" =>
  diff <= "0000000001001000";
  min <= "0001101000101011";
when "01110001" =>
  diff <= "0000000001001000";
  min <= "0001100111100011";
when "01110010" =>
  diff <= "0000000001000111";
  min <= "0001100110011011";
when "01110011" =>
  diff <= "0000000001000110";
  min <= "0001100101010100";
when "01110100" =>
  diff <= "0000000001000110";
  min <= "0001100100001110";
when "01110101" =>
  diff <= "0000000001000101";
  min <= "0001100011001000";
when "01110110" =>
  diff <= "0000000001000101";
  min <= "0001100010000011";
when "01110111" =>
  diff <= "0000000001000100";
  min <= "0001100000111110";
when "01111000" =>
  diff <= "0000000001000011";
  min <= "0001011111111010";
when "01111001" =>
  diff <= "0000000001000011";
  min <= "0001011110110111";
when "01111010" =>
  diff <= "0000000001000010";
  min <= "0001011101110100";
when "01111011" =>
  diff <= "0000000001000010";
  min <= "0001011100110010";
when "01111100" =>
  diff <= "0000000001000001";
  min <= "0001011011110000";
when "01111101" =>
  diff <= "0000000001000001";
```

```
  min <= "0001011010101111";
when "01111110" =>
  diff <= "0000000001000000";
  min <= "0001011001101110";
when "01111111" =>
  diff <= "0000000001000000";
  min <= "0001011000101110";
when "10000000" =>
  diff <= "0000000000111111";
  min <= "0001010111101110";
when "10000001" =>
  diff <= "0000000000111111";
  min <= "0001010110101111";
when "10000010" =>
  diff <= "0000000000111110";
  min <= "0001010101110000";
when "10000011" =>
  diff <= "0000000000111110";
  min <= "0001010100110010";
when "10000100" =>
  diff <= "0000000000111101";
  min <= "0001010011110100";
when "10000101" =>
  diff <= "0000000000111101";
  min <= "0001010010110110";
when "10000110" =>
  diff <= "0000000000111100";
  min <= "0001010001111010";
when "10000111" =>
  diff <= "0000000000111100";
  min <= "0001010000111101";
when "10001000" =>
  diff <= "0000000000111100";
  min <= "0001010000000001";
when "10001001" =>
  diff <= "0000000000111011";
  min <= "0001001111000110";
when "10001010" =>
  diff <= "0000000000111011";
  min <= "0001001110001010";
when "10001011" =>
  diff <= "0000000000111010";
  min <= "0001001101010000";
when "10001100" =>
  diff <= "0000000000111010";
  min <= "0001001100010101";
when "10001101" =>
  diff <= "0000000000111001";
  min <= "0001001011011011";
when "10001110" =>
  diff <= "0000000000111001";
  min <= "0001001010100010";
when "10001111" =>
  diff <= "0000000000111001";
  min <= "0001001001101001";
when "10010000" =>
  diff <= "0000000000111000";
```

```
  min <= "00010010001100000";
when "10010001" =>
  diff <= "00000000000111000";
  min <= "00010001111110000";
when "10010010" =>
  diff <= "00000000000110111";
  min <= "00010001110000000";
when "10010011" =>
  diff <= "00000000000110111";
  min <= "00010001100010000";
when "10010100" =>
  diff <= "00000000000110111";
  min <= "00010001010100001";
when "10010101" =>
  diff <= "00000000000110110";
  min <= "00010001000110100";
when "10010110" =>
  diff <= "00000000000110110";
  min <= "00010000111001000";
when "10010111" =>
  diff <= "00000000000110110";
  min <= "00010000101011100";
when "10011000" =>
  diff <= "00000000000110101";
  min <= "00010000011110000";
when "10011001" =>
  diff <= "00000000000110101";
  min <= "00010000010000110";
when "10011010" =>
  diff <= "00000000000110101";
  min <= "00010000000011100";
when "10011011" =>
  diff <= "00000000000110100";
  min <= "00001111110110010";
when "10011100" =>
  diff <= "00000000000110100";
  min <= "00001111101001010";
when "10011101" =>
  diff <= "00000000000110100";
  min <= "00001111011101100";
when "10011110" =>
  diff <= "00000000000110011";
  min <= "00001111001111010";
when "10011111" =>
  diff <= "00000000000110011";
  min <= "00001111000010100";
when "10100000" =>
  diff <= "00000000000110011";
  min <= "00001110110101110";
when "10100001" =>
  diff <= "00000000000110010";
  min <= "00001110101001000";
when "10100010" =>
  diff <= "00000000000110010";
  min <= "00001110011100100";
when "10100011" =>
  diff <= "00000000000110010";
```

```
  min <= "0000111000111111";
when "10100100" =>
  diff <= "0000000000110001";
  min <= "0000111000001110";
when "10100101" =>
  diff <= "0000000000110001";
  min <= "0000110111011100";
when "10100110" =>
  diff <= "0000000000110001";
  min <= "0000110110101011";
when "10100111" =>
  diff <= "0000000000110000";
  min <= "0000110101111010";
when "10101000" =>
  diff <= "0000000000110000";
  min <= "0000110101001001";
when "10101001" =>
  diff <= "0000000000110000";
  min <= "0000110100011001";
when "10101010" =>
  diff <= "0000000000110000";
  min <= "0000110011101001";
when "10101011" =>
  diff <= "0000000000101111";
  min <= "0000110010111001";
when "10101100" =>
  diff <= "0000000000101111";
  min <= "0000110010001010";
when "10101101" =>
  diff <= "0000000000101111";
  min <= "0000110001011011";
when "10101110" =>
  diff <= "0000000000101110";
  min <= "0000110000101100";
when "10101111" =>
  diff <= "0000000000101110";
  min <= "0000101111111101";
when "10110000" =>
  diff <= "0000000000101110";
  min <= "0000101111001111";
when "10110001" =>
  diff <= "0000000000101110";
  min <= "0000101110100000";
when "10110010" =>
  diff <= "0000000000101101";
  min <= "0000101101110011";
when "10110011" =>
  diff <= "0000000000101101";
  min <= "0000101101000101";
when "10110100" =>
  diff <= "0000000000101101";
  min <= "0000101100011000";
when "10110101" =>
  diff <= "0000000000101101";
  min <= "0000101011101010";
when "10110110" =>
  diff <= "0000000000101100";
```

```
  min <= "0000101010111101";
when "10110111" =>
  diff <= "0000000000101100";
  min <= "0000101010010001";
when "10111000" =>
  diff <= "0000000000101100";
  min <= "0000101001100100";
when "10111001" =>
  diff <= "0000000000101100";
  min <= "0000101000111000";
when "10111010" =>
  diff <= "0000000000101011";
  min <= "0000101000001100";
when "10111011" =>
  diff <= "0000000000101011";
  min <= "0000100111100001";
when "10111100" =>
  diff <= "0000000000101011";
  min <= "0000100110110101";
when "10111101" =>
  diff <= "0000000000101011";
  min <= "0000100110001010";
when "10111110" =>
  diff <= "0000000000101011";
  min <= "0000100101011111";
when "10111111" =>
  diff <= "0000000000101010";
  min <= "0000100100110100";
when "11000000" =>
  diff <= "0000000000101010";
  min <= "0000100100001010";
when "11000001" =>
  diff <= "0000000000101010";
  min <= "0000100011011111";
when "11000010" =>
  diff <= "0000000000101010";
  min <= "0000100010110101";
when "11000011" =>
  diff <= "0000000000101001";
  min <= "0000100010001011";
when "11000100" =>
  diff <= "0000000000101001";
  min <= "0000100001100010";
when "11000101" =>
  diff <= "0000000000101001";
  min <= "0000100000111000";
when "11000110" =>
  diff <= "0000000000101001";
  min <= "0000100000001111";
when "11000111" =>
  diff <= "0000000000101001";
  min <= "0000011111100110";
when "11001000" =>
  diff <= "0000000000101000";
  min <= "0000011110111101";
when "11001001" =>
  diff <= "0000000000101000";
```

```
  min <= "0000011110010100";
when "11001010" =>
  diff <= "0000000000101000";
  min <= "0000011101101100";
when "11001011" =>
  diff <= "0000000000101000";
  min <= "0000011101000100";
when "11001100" =>
  diff <= "0000000000101000";
  min <= "0000011100011011";
when "11001101" =>
  diff <= "0000000000100111";
  min <= "0000011011110100";
when "11001110" =>
  diff <= "0000000000100111";
  min <= "0000011011001100";
when "11001111" =>
  diff <= "0000000000100111";
  min <= "0000011010100100";
when "11010000" =>
  diff <= "0000000000100111";
  min <= "0000011001111101";
when "11010001" =>
  diff <= "0000000000100111";
  min <= "0000011001010110";
when "11010010" =>
  diff <= "0000000000100110";
  min <= "0000011000101111";
when "11010011" =>
  diff <= "0000000000100110";
  min <= "0000011000001000";
when "11010100" =>
  diff <= "0000000000100110";
  min <= "0000010111100010";
when "11010101" =>
  diff <= "0000000000100110";
  min <= "0000010110111100";
when "11010110" =>
  diff <= "0000000000100110";
  min <= "0000010110010101";
when "11010111" =>
  diff <= "0000000000100110";
  min <= "0000010101101111";
when "11011000" =>
  diff <= "0000000000100101";
  min <= "0000010101001001";
when "11011001" =>
  diff <= "0000000000100101";
  min <= "0000010100100100";
when "11011010" =>
  diff <= "0000000000100101";
  min <= "0000010011111110";
when "11011011" =>
  diff <= "0000000000100101";
  min <= "0000010011011001";
when "11011100" =>
  diff <= "0000000000100101";
```

```
  min <= "0000010010110100";
when "11011101" =>
  diff <= "0000000000100100";
  min <= "0000010010001111";
when "11011110" =>
  diff <= "0000000000100100";
  min <= "0000010001101010";
when "11011111" =>
  diff <= "0000000000100100";
  min <= "0000010001000101";
when "11100000" =>
  diff <= "0000000000100100";
  min <= "0000010000100001";
when "11100001" =>
  diff <= "0000000000100100";
  min <= "0000001111111101";
when "11100010" =>
  diff <= "0000000000100100";
  min <= "0000001111011000";
when "11100011" =>
  diff <= "0000000000100100";
  min <= "0000001110110100";
when "11100100" =>
  diff <= "0000000000100011";
  min <= "0000001110010001";
when "11100101" =>
  diff <= "0000000000100011";
  min <= "0000001101101101";
when "11100110" =>
  diff <= "0000000000100011";
  min <= "0000001101001001";
when "11100111" =>
  diff <= "0000000000100011";
  min <= "0000001100100110";
when "11101000" =>
  diff <= "0000000000100011";
  min <= "0000001100000011";
when "11101001" =>
  diff <= "0000000000100011";
  min <= "0000001011100000";
when "11101010" =>
  diff <= "0000000000100010";
  min <= "0000001010111101";
when "11101011" =>
  diff <= "0000000000100010";
  min <= "0000001010011010";
when "11101100" =>
  diff <= "0000000000100010";
  min <= "0000001001110111";
when "11101101" =>
  diff <= "0000000000100010";
  min <= "0000001001010101";
when "11101110" =>
  diff <= "0000000000100010";
  min <= "0000001000110010";
when "11101111" =>
  diff <= "0000000000100010";
```

```
        min <= "0000001000010000";
      when "11110000" =>
        diff <= "0000000000100010";
        min <= "0000000111101110";
      when "11110001" =>
        diff <= "0000000000100001";
        min <= "0000000111001100";
      when "11110010" =>
        diff <= "0000000000100001";
        min <= "0000000110101010";
      when "11110011" =>
        diff <= "0000000000100001";
        min <= "0000000110001001";
      when "11110100" =>
        diff <= "0000000000100001";
        min <= "0000000101100111";
      when "11110101" =>
        diff <= "0000000000100001";
        min <= "0000000101000110";
      when "11110110" =>
        diff <= "0000000000100001";
        min <= "0000000100100101";
      when "11110111" =>
        diff <= "0000000000100001";
        min <= "0000000100000100";
      when "11111000" =>
        diff <= "0000000000100000";
        min <= "0000000011100011";
      when "11111001" =>
        diff <= "0000000000100000";
        min <= "0000000011000010";
      when "11111010" =>
        diff <= "0000000000100000";
        min <= "0000000010100001";
      when "11111011" =>
        diff <= "0000000000100000";
        min <= "0000000010000001";
      when "11111100" =>
        diff <= "0000000000100000";
        min <= "0000000001100000";
      when "11111101" =>
        diff <= "0000000000100000";
        min <= "0000000001000000";
      when "11111110" =>
        diff <= "0000000000100000";
        min <= "0000000000100000";
      when "11111111" =>
        diff <= "0000000000100000";
        min <= "0000000000000000";
      when others =>
        diff <= (others => '0');
        min <= (others => '0');
    end case;
  end process;
end;
```

linear_interp.vhd [17]

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity linear_interp is
  port (in_clk : in std_logic;
      in_rand : in std_logic_vector(15 downto 0);
      in_min : in std_logic_vector(15 downto 0);
      in_diff : in std_logic_vector(15 downto 0);
      in_urn : in std_logic_vector(31 downto 0);
      out_urn : out std_logic_vector(31 downto 0);
      out_interp : out std_logic_vector(31 downto 0));
end entity linear_interp;

architecture a of linear_interp is
begin
  process(in_clk)
    variable product : std_logic_vector(31 downto 0);
    variable min_extend : std_logic_vector(31 downto 0);
  begin
    if (in_clk='1' and in_clk'event) then
           out_urn <= in_urn;
      product := in_rand * in_diff;
      min_extend(31 downto 16) := in_min;
      min_extend(15 downto 0) := (others => '0');
      out_interp <= min_extend + product;
    end if;
  end process;
end architecture a;
```

Appendix D

BRAM Based Design C++

hw.cc

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <math.h>
#include <iostream>
#include <cstdlib>
#include "iflib.h"

using namespace std;

#define NULLSPECIES 127
#define NULLRX 63
#define NMAX 127
#define MMAX 63
#define PMAX 65535
#define KMAX 65535

class CR{
  public:
      int reactants,products,fpk;
      double k;
      int renum[2],rewt[2],prnum[2],prwt[2];
};

char *memp;
int64 data;
int fd,N,M,*X,SUMS[256],TPROP[125],RXSELECT[125],ERV[125];
int seed,iterations,C,*mon,thecount;
CR *R;
double thetime;
FILE *outFile;

void init(void) {
      fd = open(DEVICE, O_RDWR);
      memp = (char *)mmap(NULL, MTRRZ, PROT_READ, MAP_PRIVATE, fd, 0);
      if (memp  == MAP_FAILED) {
            perror(DEVICE);
            exit(1);
      }
      srand(time(NULL));
}
```

```
void setSP(int index,unsigned int population){

	data.w[1] = (0x1<<27) + (index<<19);
	data.w[0] = population;
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}
}

void readSP(int index, unsigned int *dataA, unsigned int *dataB){
	data.w[1] = (0x2<<27) + (index<<19);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}

	read64(&data, memp+(1<<3));
	*dataA = (unsigned int)data.w[1];
	*dataB = (unsigned int)data.w[0];
}

void setRX(int index, int reac1, int reac2, int pro1, int pro2, int k){
	data.w[1] = (0x3<<27) + (index<<19) + (reac1<<8) + reac2;
	data.w[0] = (pro1<<24) + (pro2<<16) + k;
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}
}

void readRX(int index, unsigned int *dataA, unsigned int *dataB){
	data.w[1] = (0x4<<27) + (index<<19);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
		read64(&data, memp+(0<<3));
	}

	read64(&data, memp+(1<<3));
	*dataA = (unsigned int)data.w[1];
	*dataB = (unsigned int)data.w[0];
}

void readPROP(int index, unsigned int *dataA, unsigned int *dataB){
	data.w[1] = (0x5<<27) + (index<<19);
	write64(data, memp+(0<<3));

	read64(&data, memp+(0<<3));
	while(data.w[1]!=0x0){
```

```
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      *dataA = (unsigned int)data.w[1];
      *dataB = (unsigned int)data.w[0];
}

void readPSUM(int index1, int index2, unsigned int *dataA, unsigned int
*dataB){
      data.w[1] = (0x6<<27) + (index1<<23) + (index2<<19);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      *dataA = (unsigned int)data.w[1];
      *dataB = (unsigned int)data.w[0];
}

void setSEED(int seed){
      data.w[1] = (0x7<<27);
      data.w[0] = seed;
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }
}

void readURV(unsigned int *dataA, unsigned int *dataB){
      data.w[1] = (0x8<<27);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      *dataA = (unsigned int)data.w[1];
      *dataB = (unsigned int)data.w[0];
}

void newURV(void){
      data.w[1] = (0x9<<27);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }
}
```

```
void readPRODUCT(unsigned int *dataA, unsigned int *dataB){
      data.w[1] = (0xA<<27);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      *dataA = (unsigned int)data.w[1];
      *dataB = (unsigned int)data.w[0];
}

void readSELECTION(unsigned int *dataA, unsigned int *dataB){
      data.w[1] = (0xB<<27);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      *dataA = (unsigned int)data.w[1];
      *dataB = (unsigned int)data.w[0];
}

void readERV(unsigned int *dataA, unsigned int *dataB){
      data.w[1] = (0xC<<27);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }

      read64(&data, memp+(1<<3));
      *dataA = (unsigned int)data.w[1];
      *dataB = (unsigned int)data.w[0];
}

void initPROP(void){
      data.w[1] = (0xD<<27);
      write64(data, memp+(0<<3));

      read64(&data, memp+(0<<3));
      while(data.w[1]!=0x0){
            read64(&data, memp+(0<<3));
      }
}

void printresults(void){
      int i,j;

      for(i=0;i<125;i++){
```

```
            // UPDATE PUTATIVE TIME
            thetime+=((double)ERV[i]/536870911.0)/(double)TPROP[i];
            thecount++;


            // UPDATE SPECIES POPULATIONS
            for(j=0;j<R[RXSELECT[i]].reactants;j++){
                  X[R[RXSELECT[i]].renum[j]]-=R[RXSELECT[i]].rewt[j];
            }
            for(j=0;j<R[RXSELECT[i]].products;j++){
                  X[R[RXSELECT[i]].prnum[j]]+=R[RXSELECT[i]].prwt[j];
            }


            // PRINT TO AN OUTPUT FILE
            /*
            fprintf(outFile,"%6d %8.6lf",thecount,thetime);
            for(j=0;j<C;j++){
                  fprintf(outFile," %4u",X[mon[j]]);
            }
            fprintf("outFile,"\n");
            */
      }
}

void step(int runs){
      int i,j=0,a=0;

      while(runs>0){
            data.w[1] = (0xE<<27);
            if(runs>=125) data.w[0] = 252;
            else data.w[0] = (runs*2)+2;
            write64(data, memp+(0<<3));
            if(a==1) printresults();
            else a=1;

            read64(&data, memp+(0<<3));
            while(data.w[1]!=0x0){
                  read64(&data, memp+(0<<3));
            }

            for(i=2;i<252;i++){
                  read64(&data,memp+(i<<3));
                  TPROP[(i>>1)-1] = data.w[1];
                  ERV[(i>>1)-1] = data.w[0];

                  i++;
                  read64(&data,memp+(i<<3));
                  RXSELECT[((i-1)>>1)-1] = data.w[0];
            }
            runs -= 125;
      }
      printresults();
}


int main (int argc, char **argv)
```

```
{
      unsigned int dataA, dataB;
      int i, j, k, kl_int, MF=1, tprop, reac1, reac2, pro1, pro2;
      char modelfile[51];
      double kl=1.0,y;
      struct timeval ts,te;
      FILE *inFile;

      init();

      seed = -1;
      iterations = 1000000;

      // OPEN A FILE FOR ANY WRITING OF RESULTS
      outFile = fopen("results.txt","wt");

      // ANALYZE COMMAND LINE ARGUMENTS

      for(i=1;i<argc;i++){
            if((strcmp(argv[i],"-h")==0)||(strcmp(argv[i],"--h")==0)){
                  fprintf(stderr,"Expected usage: ./rchw [-m] [model
file] [-i] [iterations] [-s] [seed]\n");
                  exit(1);
            }
            else{
                  if(strcmp(argv[i],"-m")==0){
                        strcpy(modelfile,argv[++i]);
                  }
                  else{
                        if(strcmp(argv[i],"-i")==0){
                              iterations = atoi(argv[++i]);
                        }
                        else{
                              if(strcmp(argv[i],"-s")==0){
                                    seed = atoi(argv[++i]);
                              }
                              else{
                                    fprintf(stderr,"ERROR!  Expected
usage: ./rchw [-m] [model file] [-i] [iterations] [-s] [seed]\n");
                                    exit(1);
                              }
                        }
                  }
            }
      }

      inFile = fopen(modelfile,"r");
      while(inFile == NULL){
            printf("Please enter the name of the model file to read
from: ");
            fgets(modelfile,50,stdin);
            if(modelfile[0]==10) exit(0);
            modelfile[strlen(modelfile)-1]='\0';
            inFile = fopen(modelfile,"r");
      }
```

```
	// CLEAR BRAM
	for(i=0;i<255;i++){
		data.w[1] = 0x0;
		data.w[0] = 0x0;
		write64(data, memp+(i<<3));
	}

	// STORE INITIAL TIME OF START OF SIMULATION
	gettimeofday(&ts,NULL);


	////////////////////////////////////////////////////////////////
///////////////
	// READING MODEL FILE AND STORING VARIABLES INTO SOFTWARE
//////////////////////
	////////////////////////////////////////////////////////////////
///////////////

	// READING AND STORING SPECIES POPULATIONS
	fscanf(inFile,"%d",&N);
	if(N>NMAX){
		fprintf(stderr,"ERROR!  The number of species in this model
exceeds %d\n",NMAX);
		exit(1);
	}
	X = new int[N];
	for(i=0;i<N;i++){
		fscanf(inFile,"%d",&X[i]);
		if(X[i]>PMAX){
			X[i]=PMAX;
			fprintf(stderr,"WARNING!  Species %d exceeds maximum
and has been set to %d\n",i,PMAX);
		}
	}

	gettimeofday(&ts,NULL);

	// READING AND STORING REACTION EQUATIONS
	fscanf(inFile,"%d",&M);
	if(M>MMAX){
		fprintf(stderr,"ERROR!  The number of reactions in this
model exceeds %d\n",MMAX);
		exit(1);
	}
	R = new CR[M];
	for(i=0;i<M;i++){

		// READING EACH REACTION'S REACTANTS
		fscanf(inFile,"%d",&R[i].reactants);
		k=0;
		for(j=0;j<R[i].reactants;j++){
			fscanf(inFile,"%d",&R[i].rewt[j]);
			k+=R[i].rewt[j];
			fscanf(inFile,"%d",&R[i].renum[j]);
		}
		for(j;j<2;j++){
			R[i].rewt[j]=0;
```

```
                  R[i].renum[j]=NULLSPECIES;
            }
            if(k>2){
                  fprintf(stderr,"ERROR!  The number of reactants in
reaction %d exceeds 2\n",i);
                  exit(1);
            }

            // READING EACH REACTION'S PRODUCTS
            fscanf(inFile,"%d",&R[i].products);
            k=0;
            for(j=0;j<R[i].products;j++){
                  fscanf(inFile,"%d",&R[i].prwt[j]);
                  k+=R[i].prwt[j];
                  fscanf(inFile,"%d",&R[i].prnum[j]);
            }
            for(j;j<2;j++){
                  R[i].prwt[j]=0;
                  R[i].prnum[j]=NULLSPECIES;
            }
            if(k>2){
                  fprintf(stderr,"ERROR!  The number of products in
reaction %d exceeds 2\n",i);
                  exit(1);
            }

            // READING EACH REACTION'S K
            fscanf(inFile,"%lf",&R[i].k);
            y = R[i].k - (int)R[i].k;
            if((y>0) && (y<kl)) kl=y;
      }

      // READING SPECIES TO BE MONITORED
      fscanf(inFile,"%d",&C);
      mon = new int[C];
      for(i=0;i<C;i++){
            fscanf(inFile,"%d",&mon[i]);
      }

      // DETERMING MULTIPLICATION FACTOR (MF) TO CHANGE K VALUES TO
FIXED POINT
      if(kl < 1){
            MF = 10000000;
            if(kl < 0.0000001){
                  MF = (int)(1.0/kl);
            }
            kl_int = (int)(kl * MF);
            if((int)(kl * MF * 10)%10 >= 5) kl_int += 1;
            for(i=0;i<6;i++){
                  if(kl_int %10 > 0) break;
                  MF /= 10;
                  kl_int /= 10;
            }
      }

      // UPDATE FIXED POINT K VALUE FOR EACH REACTION
      for(i=0;i<M;i++){
```

```
            R[i].fpk = (int)(R[i].k * MF);
            if((int)(R[i].k * MF * 10) % 10 >= 5) R[i].fpk += 1;
            if(R[i].fpk > KMAX){
                  fprintf(stderr,"ERROR!  Fixed point rate constant of
reaction %d exceeds %d\n",i,KMAX);
                  exit(1);
            }
      }


      ////////////////////////////////////////////////////////////////
///////////////
      // READY TO INTERFACE WITH FPGA
////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////
///////////////

      setSEED(seed);
      // Set species populations
      for(i=0;i<N;i++){
            setSP(i,X[i]);
      }
      for(i;i<NULLSPECIES;i++){
            setSP(i,0);
      }
      setSP(NULLSPECIES,1);

      // Set reaction equations
      for(i=0;i<M;i++){
            if(R[i].rewt[0]==2){
                  reac1 = R[i].renum[0];
                  reac2 = R[i].renum[0];
            }
            else{
                  reac1 = R[i].renum[0];
                  reac2 = R[i].renum[1];
            }
            if(R[i].prwt[0]==2){
                  pro1 = R[i].prnum[0];
                  pro2 = R[i].prnum[0];
            }
            else{
                  pro1 = R[i].prnum[0];
                  pro2 = R[i].prnum[1];
            }
            setRX(i,reac1,reac2,pro1,pro2,R[i].fpk);
      }
      for(i;i<=NULLRX;i++){
            setRX(i,NULLSPECIES,NULLSPECIES,NULLSPECIES,NULLSPECIES,0);
      }

      step(iterations);

      gettimeofday(&te,NULL);
      printf("te = %6d.%6d\nts =
%6d.%6d\n",te.tv_sec,te.tv_usec,ts.tv_sec,ts.tv_usec);
```

```
	for(i=0;i<N;i++){
		readSP(i,&dataA,&dataB);
		printf("Species[%d] = %10d\n",i,dataB);
	}

	munmap(memp, MTRRZ);

	return 0;
}
```

Appendix E

SBML Models

SBML Content Outline

```
[Number of species]
[Population of each species, all separated by a space]
[Number of reactions]
[Reaction equations defined in form outlined below]
[# Reactants] [Weight] [Index] [# Products] [Weight] [Index] [k]
[Number of species to be monitored]
[Indices of species populations to be monitored]
```

When defining a reaction equation, “Index” refers to index of a species involved in a reaction equation while “Weight” refers to the number of that species acting as a reactant or product in a given reaction equation. A weight and an index are defined for the number of reactants as well as the number of products.

Protein Dimerization [18]

```
8
1 1 0 0 0 0 0 0
13
1 1 0 2 1 0 1 2 0.01
1 1 2 0 6e-3
1 1 2 2 1 2 1 4 3e-2
1 1 4 0 4e-4
2 1 6 1 1 1 1 7 0.0016
1 1 7 2 1 1 1 6 0.2
1 1 1 2 1 1 1 3 0.002
1 1 7 2 1 7 1 3 0.1
1 1 3 0 6e-3
1 1 3 2 1 3 1 5 3e-2
1 1 5 0 4e-4
1 2 4 1 1 6 0.016
1 1 6 1 2 4 1
8
0 1 2 3 4 5 6 7
```

Original Tuberculosis SBML [18]

```
17
10 0 0 10 0 0 10 20 0 0 0 0 0 0 1 0 1
23
1 1 16 2 1 0 1 16 10
2 1 0 1 3 2 1 3 1 1 16
2 1 0 1 4 2 1 4 1 1 32
2 1 0 1 5 2 1 5 1 1 16
2 1 1 1 3 2 1 3 1 2 0.6
2 1 1 1 4 2 1 4 1 2 0.6
2 1 1 1 5 2 1 5 1 2 0.78
1 1 2 0 4
1 1 3 1 1 4 100
1 1 4 1 1 3 1
1 1 4 1 1 5 0.5
1 1 5 1 1 4 10
2 1 6 1 7 1 1 8 1
1 2 1 1 1 13 10
1 1 13 1 2 1 10
2 1 13 1 14 1 1 15 5
1 1 15 2 1 13 1 14 10
2 1 15 1 9 2 1 8 1 15 10
1 1 8 1 1 9 10
1 1 9 1 1 10 4
1 1 10 1 1 11 6
1 1 11 3 1 12 1 6 1 7 1
1 1 12 0 100
17
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

Modified Tuberculosis SBML [18]

```
18
10 0 0 10 0 0 10 20 0 0 0 0 0 0 1 0 1 0
24
1 1 16 2 1 0 1 16 10
2 1 0 1 3 2 1 3 1 1 16
2 1 0 1 4 2 1 4 1 1 32
2 1 0 1 5 2 1 5 1 1 16
2 1 1 1 3 2 1 3 1 2 0.6
2 1 1 1 4 2 1 4 1 2 0.6
2 1 1 1 5 2 1 5 1 2 0.78
1 1 2 0 4
1 1 3 1 1 4 100
1 1 4 1 1 3 1
1 1 4 1 1 5 0.5
1 1 5 1 1 4 10
2 1 6 1 7 1 1 8 1
1 2 1 1 1 13 10
1 1 13 1 2 1 10
2 1 13 1 14 1 1 15 5
1 1 15 2 1 13 1 14 10
2 1 15 1 9 2 1 8 1 15 10
1 1 8 1 1 9 10
1 1 9 1 1 10 4
1 1 10 1 1 11 6
1 1 11 2 1 12 1 17 1
1 17 2 1 6 1 7 655
1 1 12 0 100
17
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

Vita

Brandon Parks Thurmon was born on August 1, 1979 in Dyersburg, Tennessee. He was raised in Dyersburg, Tennessee and graduated from Dyersburg High School in 1997. Brandon enrolled in the Electrical Engineering program at the University of Tennessee, Knoxville in the fall of 1997. During his undergraduate studies at the University of Tennessee, he participated in the university's cooperative engineering program. He worked at Computational Systems Incorporated in Knoxville, Tennessee for four semesters. In May of 2002, he graduated from the University of Tennessee with a Bachelor of Science degree in Electrical Engineering. Brandon graduated once again from the University of Tennessee, Knoxville in May of 2003 with a Bachelor of Science degree in Computer Engineering. Following a brief hiatus, Brandon returned to the University of Tennessee, Knoxville in the spring of 2004 to attain his graduate degree. Brandon graduated in August of 2005 with a Master of Science degree in Electrical Engineering.

Brandon will be relocating to St. Louis, Missouri to pursue a career in the engineering field.