Stochastic chemical kinetics on an FPGA: Bruce R Land

Introduction

As you read this, there are thousands of chemical reactions going on in your body. Some are very fast, for instance, the binding of neurotransmitters in your brain occurs on a time scale of microseconds, while protein production has a time scale of seconds or minutes. Understanding physiology requires understanding how chemicals react and how fast each reaction is. *Chemical kinetics* is the name given to the study of reaction speed. But why would an article on chemical kinetics appear in this electronics magazine? The short explanation is that hardware instantiated on an FPGA can accelerate the modeling of some reactions and is an interesting example of parallel computation. The details follow.

To make things specific, let's consider one reaction, the breakdown of starch into sugar by the enzyme amylase. Amylase is present in saliva and is the reason that rice or potato becomes sweeter as you chew, as the starch is converted to sugar. An enzyme is a chemical which accelerates the reaction rate of another chemical, so you can immediately see that this is a nontrivial (and nonlinear) example. The reaction is often represented as

amylase + starch $\leftrightarrow$ (amylase- bound-to-starch) $\rightarrow$ amylase + sugars

Which should be read as: Amylase and starch in water solution react to form a loosely bound entity, which clips the starch apart into sugars. The sugars then fall off the amylase, so that the amylase is free to react with more starch. The bidirectional arrow representing the binding also suggests that the combination of amylase- bound-to-starch can also just fall apart without conversion to sugar. However, the second reaction, where the bound combination falls apart to the sugar products never reverses. This formulation is known as a Michaelis-Menten description of the reaction.

There are three reaction rates associated with this reaction; the binding rate of amylase to starch, the unbinding rate of the combination back to amylase and starch, and the reaction rate to sugar. Let us now consider what determines to reaction rates. Since the chemicals are in a water solution, the individual molecules are dispersed and have to physically collide with each other to react. The rate at which they collide is partly determined by how many there are in a given volume. Doubling the concentration doubles the chances of collision and therefore the reaction rate. The rate is also sensitive to all kinds of influences from temperature, pH, etc, but we will model all those effects as a *rate constant* independent of the concentration of the chemicals.

Now we need a bit of notation. We will use square brackets to mean *concentration of* x, written [x]. We will call the rate constant of the binding reaction k1, the unbinding rate constant k2,

and the conversion rate constant k3. The proportionality of reaction rate with concentration implies that for the first reaction the rate is k1*[amylase]*[starch]. The classical way of solving this system is to convert the rates to differential equations, then solve the set of differential equations. Quite often, including this example, we cannot solve the equations analytically, but need to simulate the solutions. For these reactions we can write

d [amylase]/dt = - k1*[amylase]*[starch] + (k2+k3)*[amylase- bound-to-starch]

d[sugars]/dt = k3*[amylase- bound-to-starch]

Hidden in these equations is the assumption of differential smoothness in concentration. For huge numbers of molecules (like the number of water molecules in a teaspoon) differential smoothness makes sense because removing one molecule is completely undetectable. But what if the volume of interest is a single cell that might have only a few copies of a molecule in a tiny volume? The concentration is still high, so the reaction goes quickly, but the concentration can only change in units of one molecule and cannot change smoothly, so the differential equation approach fails. Instead, you have to treat each molecule as having some probability of reacting. This finally leads us back to the topic of this article: Using an FPGA to accelerate probabilistic simulations of chemical reactions by counting reaction events based on random number generation.

The algorithm

The approach we will take is to step back to a more fundamental level than differential equations and treat each potential reaction event as a random occurrence, biased by chemical concentration and rate constants. But to emphasize that chemicals are discrete molecules, we will replace concentrations by *number* of molecules, with notation Nmolecule. During any small interval of time we can calculate the probability of the reaction occurring as the product of rate constants and number of molecules, just as before. So we *could* map reaction calculations to parallel multipliers on the FPGA, then ask if the product, say, k1*Namylase*Nstarch is greater than a uniform random number and allow one reaction to occur if it is. However hardware multipliers tend to be limited in number on FPGAs, relative to general logic elements so we need to simplify the calculation. As explained in Salwinski and Eisenberg , if we compare each number of molecules (and rate constant) to a uniformly distributed random number then the probability of (Nchemical>random-number) is proportional to number (or rate constant). Since each chemical number, rate constant and random number is independent, the probability of the product of the concentrations and rate constants be above some value is equal to the probability that
(k1>rand1) && (Namylase>rand2) && (Nstarch>rand3)
where && is the usual logical *and* operation. Thus we have replaced expensive multiplies with

cheap, fast random number generation and logical operations. Each term of the differential equation is replaced with these stochastic operations and the left hand side rate term is replaced with an increment/decrement or no change operation.

We still need to generate random numbers, without using multipliers. It is possible to generate very high quality pseudorandom numbers using a fairly long shift register with exclusive-or feedback from the appropriate stages to stage 1. You can find out how to do this by searching for *linear feedback shift register* (for instance *en.wikipedia.org/wiki/Linear_feedback_shift_register**)***. Hoogland et.al. show how to construct a high quality 16-bit or 32-bit pseudorandom number in one or two shift cycles by folding a long (127-bit) shift register into 16 sections and shifting all of them at once. The Verilog code will give details.

Some care has to be taken that the actual reaction rates are not too large or too small. Too large and more than one or two reactions may happen per time step which makes the process of computing the probabilities harder. Too small and the simulation takes too long. I modified the system used by in Salwinski and Eisenberg to allow up to two reactions at each time step, whereas they used only one. Limiting the actual reaction rate to an average probability of no more than 0.085/step keeps the error due to missed events below 0.01% (based on a Poisson distribution). If you can relax the missed events on 0.1% then a probability of 0.15 may be used. Computing the actual reaction rate means multiplying out the rate constant, and the one or two chemical concentrations (as a fraction of 2^16, since I used 16-bit concentrations). For instance, a first order reaction with rate constant 4096 and concentration 8192 would have a reaction probability of (4096/2^16)*(8192/2^16) or (1/16)*(1/8)=0.0078.

Hardware organization

The hardware on the FPGA is organized into several modules. System control is a state machine which sequences through eight states. In the first state, all random numbers are generated and reaction results computed. In the next seven states, the reaction results are added/subtracted to the various chemical number counters. The logical result is a cycle of computing reactions, then updating chemical numbers on each time step. There is a chemical module instantiated for each different chemical and a reaction module for each different reaction path. All of the reaction modules and all of the chemical modules compute their contribution to the current time step in eight clock cycles, independent of the number of chemicals or reactions.

The following code fragment shows the structure of the enzyme reaction when it is converted into Verilog code. There is a module defined for each chemical and one for each reaction. Each chemical is defined by one hardware module which outputs the current number of molecules for the chemical. Inputs include an initial concentration, slots for up to six

increment/decrement commands (from reaction modules), the reaction clock, reaction state, and a reset command. Internally, the chemical module is a state machine which uses the increment/decrement commands to compute the updated number of molecules at each time step.

Each reaction is defined by one hardware module which outputs increment/decrement commands to feed back to chemical modules. Inputs are the number of molecules of one or two chemicals, a rate constant, the reaction clock, reaction state, a random number seed and a reset command. The random number seed should be distinct for each different reaction module. Internally, the reaction module computes six random numbers in parallel (three for each of two possible reactions at each time step), compares them to the molecular numbers and rate constant, then determines if zero, one or two reactions actually occurs by performing the logical and operation described above.

Code sidebar caption: This Verilog code defines the reaction explained in the introduction for simulating a enzymatic reaction. Three chemicals and three reactions are defined. The Full Verilog code contains the modules *chemical* and *reaction* modules. See http://instruct1.cit.cornell.edu/courses/ece576/Chemical_Simulation/index.html

```
// Michaelis and Menten ////////////////////////////////
//////////////////////////////////////////////////////
// define A + E <-> AE -> S + E  (enzyme reaction)
//////////////////////////////////////////////////////
wire [15:0] A, S, AE, E ; // concentrations
// concentration inc/dec from reactions
wire [2:0] AtoAE_inc, AtoAE_dec,
           AEtoA_inc, AEtoA_dec,
           AEtoS_inc, AEtoS_dec;

// Handy constants used for unused inputs to modules
parameter no_chem = 16'hffff, no_inc = 3'b000 ;

// Read this as:
// For chemical A the initial condition is 240 molecules.
// When A is converted to AE, decrement A.
// When AE is converted back to A, increment A.
// Four increment inputs are not used.
// All chemical modules need the state variable, reaction clock and reset.
chemical chem_A( A, 16'd240,
                    AtoAE_dec, AEtoA_inc,
                    no_inc, no_inc,
                    no_inc, no_inc,
```

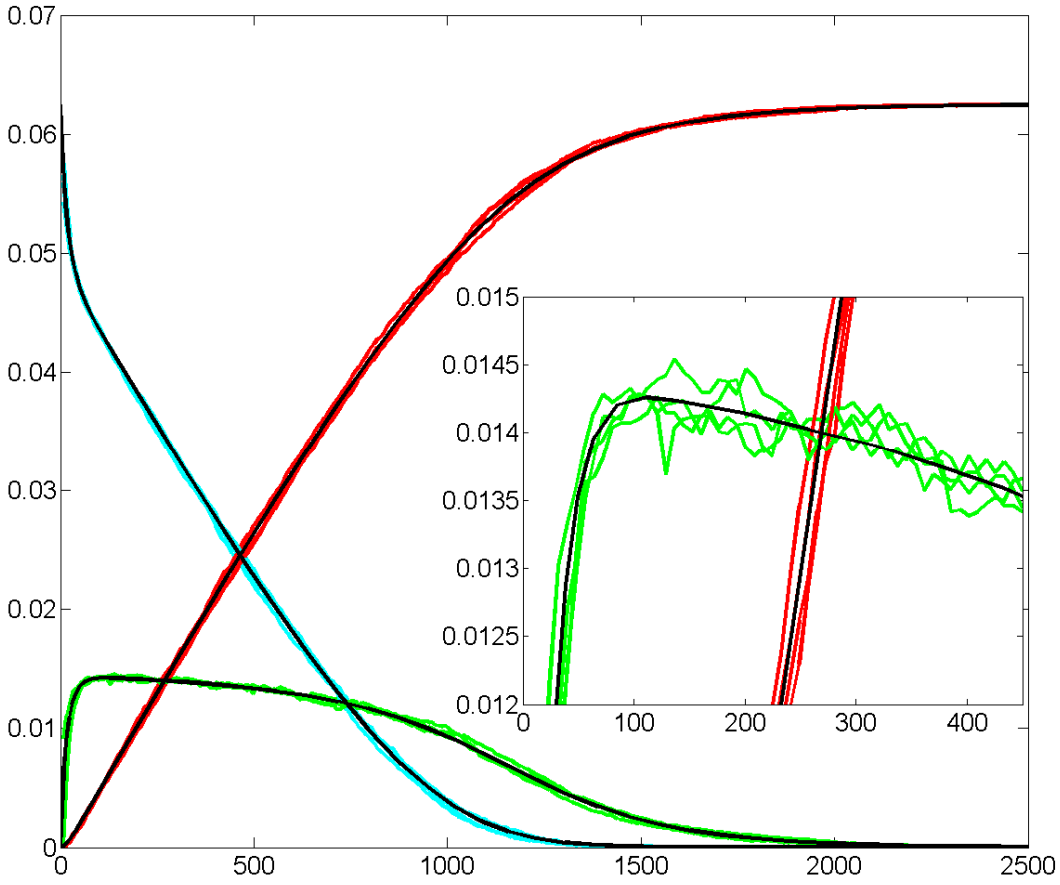
```
                        state, reaction_clock, reset);

chemical chem_S( S, 16'h0000,
                        AEtoS_inc, no_inc,
                        no_inc, no_inc,
                        no_inc, no_inc,
                        state, reaction_clock, reset);

chemical chem_E( E, 16'd60,
                        AtoAE_dec, AEtoA_inc, AEtoS_inc, no_inc,
                        no_inc, no_inc,
                        state, reaction_clock, reset);

chemical chem_AE( AE, 16'h0000,
                        AtoAE_inc, AEtoA_dec, AEtoS_dec, no_inc,
                        no_inc, no_inc,
                        state, reaction_clock, reset);

// define the forward and backward reactions.
// inc/dec output signals are nonzero  if the reaction occurs.
// unused concentration inputs should be set to no_chem=16'hffff.
// Read this as:
// If the reaction of A+E to AE occurs, set the inc/dec lines to nonzero.
// The input chemicals are A and E, with rate constant 16'hffff
// All reaction modules require state, clock, reset and a unique seed.
reaction AtoAE(AtoAE_inc, AtoAE_dec, A, E, 16'hffff,
                        state, reaction_clock, reset, 128'haaaaaaaa54555555+seed_offset);

reaction AEtoA(AEtoA_inc, AEtoA_dec, AE, no_chem, 16'h0010,
                        state, reaction_clock, reset, 128'haaaaaaaa55555555+seed_offset);

reaction AEtoS(AEtoS_inc, AEtoS_dec, AE, no_chem, 16'd256,
                        state, reaction_clock, reset, 128'haaaaaaaa53555555+seed_offset);
```

I wanted to be able to visualize the reactions, so I wrote a time-series VGA display module which takes three different data inputs and plots them as they are calculated in three different colors. The VGA interface used dual-ported memory on the FPGA as display memory and operates completely in parallel with the reaction state machine. The screen refresh side of the VGA controller read from memory (Altera M4K blocks) when it needed to draw the screen, while the data formatting side of the controller wrote to display memory in a state machine. Since it takes a lot of small time steps to compute some reactions, not every reaction time step is plotted on the VGA. The state machine handles the display time increment.

Hardware performance and comparison with software algorithms

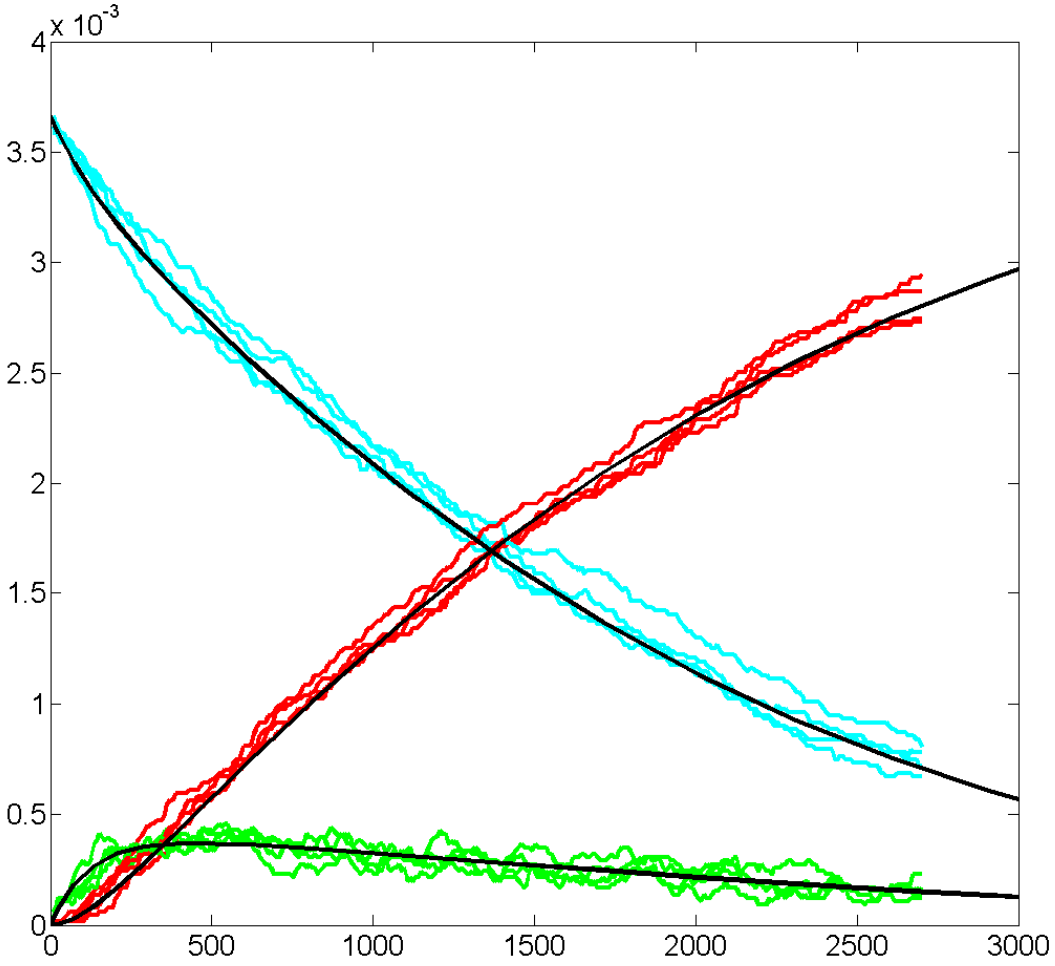
The reaction simulation results from the FPGA were compared to both the same stochastic algorithm coded in Matlab and to a differential equation formulation coded in Matlab (A. van Oudenaarden, MIT). Two chemical systems tested were the Michaelis-Menten formulation of enzymatic action explained in the introduction and a nonlinear oscillator known as the *Oregonator* model because it was originally developed at the University of Oregon by Field and Noyes.

The Michaelis-Menten formulation is a good test case because it is familiar to all biochemists and includes a nonlinear term. Figure 1 shows a typical result comparing the FPGA hardware with Matlab differential equation code. Since the numbers of molecules is fairly large in this example (4096 substrate, 1024 enzyme) we expect the FPGA stochastic simulation hardware to approximate closely the differential equation solution because the stochastic RMS variation is proportional to the square-root of the number of molecules. So the RMS variation should be around sqrt(1024)/1024 , or about 3%. The inset suggests a few percent RMS variation near the peak of the bound enzyme. The close correspondence between the Matlab and FPGA results suggests that the FPGA parallel design is correct.



Caption Figure 1. Michaelis-Menten kinetics were used to test the stochastic solver. The cyan curve is the substrate number, red curve is the product, and lower curve the concentration of bound enzyme. The black lines are the values computed by a Matlab differential equation solver. The inset shows the stochastic variability near the bound enzyme peak.

Figure 2 shows the effect of dropping the number of molecules down to only 60 enzyme molecules. At the peak, only about 25 enzyme molecules are bound and the statistical fluctuations are sqrt(25)/25, or about 20%, which are clearly visible. In this case, the averaging properties of the differential equation become obvious. In real cells, the number of enzyme molecules is often quite small, and the reality of stochastic variation directly effects cell operation and is more 'real' than the differential equation average.



Caption Figure 2. Michaelis-Menten kinetics at lower concentration. The cyan curve is the substrate number, red curve is the product, and green curve the concentration of bound enzyme. The black lines are the values computed by a Matlab differential equation solver. The stochastic variation is quite noticeable, and in some ways is a better representation of what goes on in individual biological cells.

The Oregonator is a good test case because it a classic example of a *stiff system*. A stiff system is one in which there are a large range of characteristic reaction rates. This feature makes the system harder to solve with a differential equation solver, but it works well with a stochastic solver . This system was devised by Field and Noyes at the University of Oregon (1974) and used by Gillespie (1977) as a test case for an exact stochastic simulator method. The reaction scheme is:

X1 + Y2 → Y1 (rate constant c1)
Y1 + Y2 → Z1 (rate constant c2)
X2 + Y1 → 2Y1 + Y3 (rate constant c3)
Y1 + Y1 → Z2 (rate constant c4)
X3 + Y1 → Y2 (rate constant c5)
The X's represent large pools of chemical and do not change concentration during simulation, so they are just constants. The Z's are reaction products which are not reused and therefore do not need to be modeled. The three Y molecule numbers are shown in figure 3. The results of seven stochastic runs are plotted. You can see that there is considerable variability, centering around the differential equation solution (black lines).

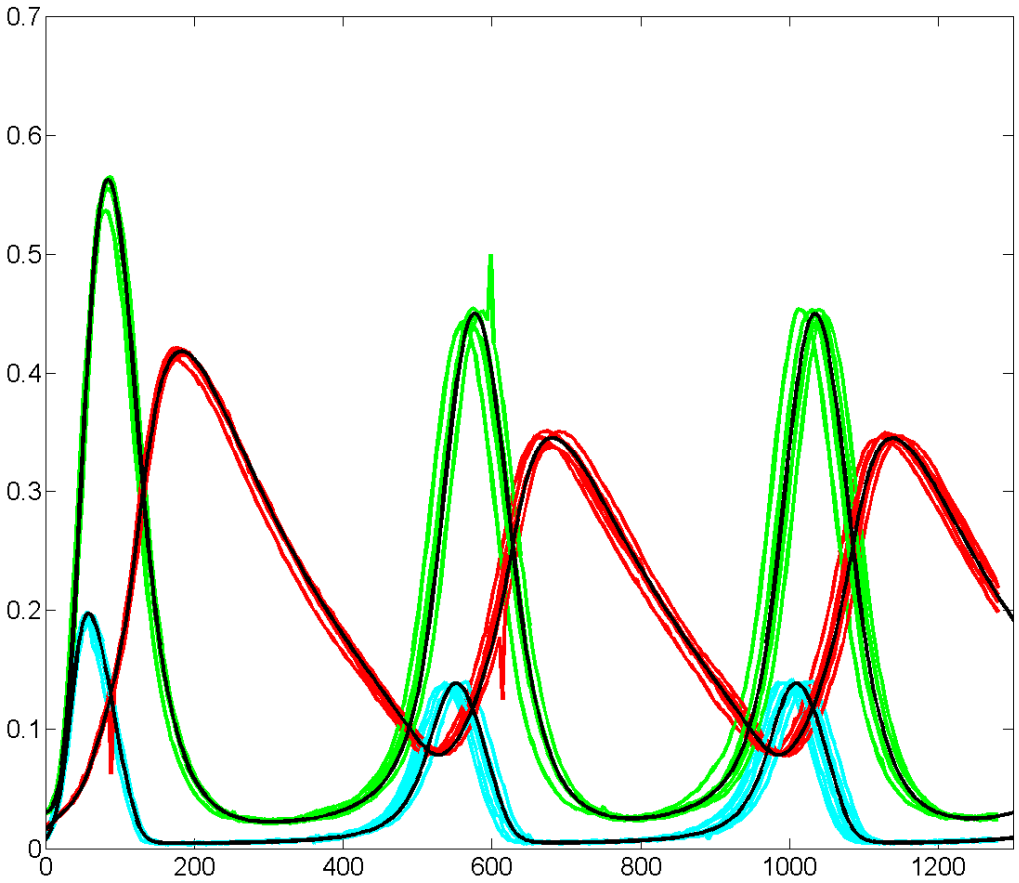


Figure 3. Oregonator kinetics calculated by Matlab ODE solver (black lines) and the FPGA stochastic solver. There are three communication glitches on the traces. Cyan is Y1, red is Y2 and green is Y3.

Conclusions

Even though the clock rate of the FPGA is rather low compared to a PC, the computational rate of the FPGA stochastic simulation is faster than the PC because so many operations can be carried out in parallel. In the case of the Oregonator simulation, thirty 16-bit random numbers are computed in one cycle to support computation of the reaction and three chemical concentrations are updated in seven cycles. The update rate is independent of the number of chemicals or reactions (up to the size limit of the FPGA), so bigger models show more speedup over the PC solution. The Matlab stochastic simulator I wrote took 870 seconds to run on my desk machine (3.2 GHz Core Duo with 8 Gbyte memory) and 8 seconds to run on the FPGA, a factor of 100 speed up.

The Oregonator model (with serial readout and VGA display) uses about 15% of the CycloneII FPGA on the Altera DE2 educational board. The VGA alone requires about 2% and the serial readout module is of negligible size.

Full Verilog code and further details are available at
http://instruct1.cit.cornell.edu/courses/ece576/Chemical_Simulation/index.html

References

Salwinski L, Eisenberg D., *In silico simulation of biological network dynamics*.
Nature Biotechnology, 2004 Aug;22(8):1017-9

A. Hoogland, J. Spaa, B. Selman and A. Compagner, *A special-purpose processor for the Monte Carlo simulation of ising spin systems*, Journal of Computational Physics, Volume 51, Issue 2, August 1983, Pages 250-260

A. van Oudenaarden, 7.32/7.81J/8.591J Systems Biology, MIT, September 2009
http://web.mit.edu/biophysics/sbio/PDFs/L2_notes.pdf

R. J. Field, R. M. Noyes, *Oscillations in Chemical Systems IV. Limit cycle behavior in a model of a real chemical reaction*, J. Chem. Phys. 60(1974)1877-84.

D. Gillespie, *Exact Stochastic Simulation of Coupled Chemical Reactions*, Journal of Physical Chemistry, No. 81, pp. 2340-2361, 1977.