

# Reconfigurable Molecular Dynamics Simulator

Navid Azizi, Ian Kuon, Aaron Egier,  
Ahmad Darabiha, and Paul Chow

Department of ECE, University of Toronto, Toronto, Ontario Canada  
{nazizi,ikuon,aegier,ahmadd,pc}@eecg.utoronto.ca

## ABSTRACT

Current high-performance applications are typically implemented on large-scale general-purpose distributed or multi-processing systems often based on commodity microprocessors. Field-Programmable Gate Arrays (FPGAs) have now reached a level of sophistication that they too could be used for such applications.

In this paper we explore the feasibility of using FPGAs to implement large-scale application-specific computations by way of a case study that implements a novel Molecular Dynamics system. The system has been designed such that it is scalable and parallelizable. On the Transmogripher 3 (TM3), the system performs calculations on an 8,192 particle system in 37 seconds at 26MHz. This implementation shows that by scaling to more modern parts running at 100MHz, a speedup of over 20x can be achieved compared to a state-of-the-art microprocessor. This can also be achieved at less cost, using less power and taking less space than a standard microprocessor-based system, while maintaining the computational precision required.

## 1. INTRODUCTION

The goal of this work is to explore the feasibility of using Field-Programmable Gate Arrays (FPGAs) to implement large-scale processing systems that can compete with current high-end computing technology.

As a way to test our hypothesis, we implement a key function used to solve Molecular Dynamics problems and show that FPGAs are a viable technology for the high-performance application space.

### 1.1 Application-Specific Processors

Ever since computing was invented, there has always been the challenge of designing a computer to go faster and solve the next larger problem. One approach is to build systems that are programmable using a programming language. Early supercomputers were typically vector processors, such as the Cray-1 [1], evolving to the large-scale distributed and multiprocessing systems of today. An alternate approach, for well-defined problems, is to build a highly-customized machine that can solve one specific problem very well, such as spacial-lattice computations [2] where a custom integrated circuit was built. These *Application-Specific Processors* (ASP) use architectures that take significant advantage of knowledge about the problem and typically have little programmability.

Whenever developing an ASP, there are two key risks to consider:

**Development Cycle** By the time the ASP is completed, will it still have a significant advantage over the latest microprocessor-based software solution?

**Correct Solution** Is the chosen solution the best solution?

Many ASPs have failed because they did not provide enough advantage over a software solution. Long development times meant that when the ASP was ready to be used, processor speeds had increased enough such that it was not worth it to use the ASP. It was easier to port the existing software to the faster processor. The construction of an ASP should only be considered if the perceived gain is significantly better than what microprocessors could achieve in the same time frame.

An ASP also typically uses a hardwired architecture that is tuned for a specific algorithm. This means that it is unlikely that the algorithm can be modified if changes are desirable. As a result, there has not been significant activity building application-specific computing architectures, particularly for high-end applications.

Building ASPs means that there must be significant knowledge about the application. In many modern problems, the applications come from areas of study that are not readily understood by hardware architects and designers. There would need to be significant collaboration between disciplines to develop a good solution. For example, to develop a system for modeling biomolecular systems, there would need to be biochemists and computer architects working together.

### 1.2 Hypothesis

The time has come to revisit the development of ASPs. Modern FPGAs have now reached the level of sophistication that significant hardware computing functions, or engines, can be incorporated into a single FPGA. FPGAs offer significantly reduced development risk and the flexibility to modify the design of a compute engine if desired, thus addressing the two key risks associated with ASPs. To get increased parallelism, many FPGAs can be connected together in a manner desirable for the application.

Although it may seem that an FPGA cannot compete with a multi-Gigahertz processor in performance, it is important to realize that for many types of problems the memory is the bottleneck, not the processor speed. With a customized memory system and the ability to offer more parallelism, it should be possible to achieve performance benefits over a software solution on a fast microprocessor. In addition, there will be advantages in cost, power, and physical size. The advantage of an application-specific system is that

it is more customized to the application and therefore is unlikely to need the sophisticated features of a general-purpose computing system. For example, an expensive cache hierarchy would not be required, saving significant hardware, which means less cost, power, and size.

### 1.3 Testing the Hypothesis

To test the viability of using FPGAs for high-performance computing applications we chose the application called Molecular Dynamics (MD). In MD simulations, the behavior of molecular systems are modeled at the atomic level simply by computing the forces on each atom and integrating the classical equations of motion [3]. By doing a real MD implementation, the goal is to expose the issues that will be faced when building a large FPGA-based ASP.

Molecular Dynamics has been in development since the 1950's [3]. Simulations have progressed from simple modeling of liquids to areas such as defect analysis. Currently, a major area of interest is biomolecules with a goal of simulating molecular activities such as protein folding. To achieve such goals, the computational requirements have grown enormously. Computation on a single processor is no longer sufficient and massively parallel systems are being attempted. This makes MD a suitable candidate for testing the feasibility of developing FPGA-based, application-specific solutions to high-performance computing problems.

More specifically, one of the compute-intensive aspects of MD will be designed in VHDL and implemented on the TM3 development platform. The goal will be to construct a hardware-based MD engine that is easily parameterized in terms of calculation constants and precision while also providing the potential benefits of hardware acceleration.

The TM3 has become somewhat dated compared to current FPGA technologies, so there is no expectation for speed-up. However, it will be shown that enough insight can be gained to be able to confidently predict the performance of a system based on more recent technology.

For this project to be a success, it is also imperative that the results from the hardware computations be as good as the standard software implementations. Accordingly, another goal of the project is to match the results with software MD simulations. At the same time, this goal should be achieved by using the minimal amount of hardware. Fixed-point arithmetic has been used throughout this project.

In summary, the overall goal is to demonstrate that it is feasible to implement an FPGA-based application that can be faster, lower power, lower cost, less in size, and produces results that are at least as good as what can be done with current high-end computing systems.

### 1.4 Overview

Some background for understanding MD simulations and the hardware system that was used will be provided in Section 2. Section 3 will describe each of the hardware modules that make up the design and Section 4 will outline the reconfigurability of the system. Section 5 will outline the results and the validation strategy of the MD system itself. In Section 6, we show how the designed system is scalable and parallelizable and can produce large speed-up given a better memory organization. Section 7 gives a comparison of this system versus the more typical microprocessor-based high-performance computing alternatives. Finally, Section 8 concludes.

## 2. BACKGROUND

In this section, a brief overview of MD and the calculations required are given followed by a description of previous systems built to accelerate MD computations. A very brief description of the TM3 platform is also given.

### 2.1 MD Background

MD is a simulation technique for modeling the motion and interaction of atoms or molecules using the equations of classical Newtonian mechanics. For every particle in the system, the forces acting upon it are computed. Given the forces, acceleration, velocity and distance moved can be computed for a timestep typically on the order of 1 femtosecond.

There are many types of forces that can act on a particle. These can be divided into the bonded and non-bonded forces. The bonded forces are an  $O(n)$  problem and will not be considered here. The non-bonded interactions are an  $O(n^2)$  problem and take the bulk of the computation time. The most common non-bonded interaction is modeled by the Lennard-Jones (LJ) potential [3]. The LJ potential is defined by the following function:

$$\phi_{LJ}(r) = 4\epsilon \left[ \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^6 \right] \quad (1)$$

where  $\epsilon$  and  $\sigma$  are particle specific constants and  $r$  is the separation between the two interacting particles. Using this expression for potential, the force experienced between the particles can be calculated as :

$$F = -\nabla_r \phi_{LJ}(r) \quad (2)$$

At extremely close ranges, the force is repulsive as the  $r^{-12}$  term dominates. This represents the repulsion between electron clouds when the particles are near each other. As the distance between particles is increased, the force becomes attractive and the potential roughly approximates the Van der Waal's forces.

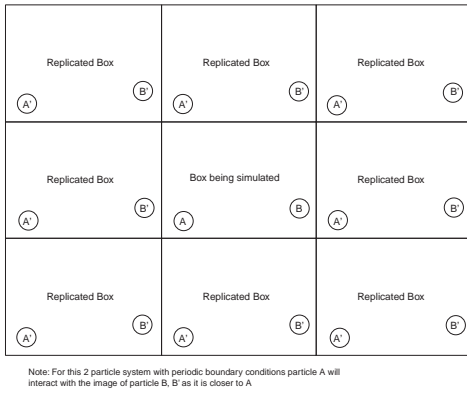
This potential,  $V$ , is calculated for every possible pair of interacting particles as in

$$V(r_1, \dots, r_N) = \sum_i \sum_{j < i} \phi_{LJ}(|\vec{r}_i - \vec{r}_j|) \quad (3)$$

which then is used to obtain the net force acting on any given particle,  $F_i = -\nabla_{r_i} V(r_1, \dots, r_N)$ . For the system described in this paper, the LJ force will be the only force considered, which is adequate for many studies. The other important  $O(n^2)$  force is the Coulombic force and a more in depth examination can be found in Allen and Tildesley [4].

The number of particles simulated is also an issue in MD simulations. Simulating a liter of ideal gas containing about  $2.7 \times 10^{22}$  particles is clearly not tractable. Instead, a technique called periodic boundary conditions is used. When assuming periodic boundary conditions, the region being simulated is replicated in every dimension as depicted in Figure 1 for two dimensions. Through this replication, the dominance of surface effects is eliminated and a comparatively small number of particles in a larger system can be simulated.

The one caveat regarding the use of periodic boundary conditions is that the volume under consideration must be sufficiently large so that each particle has effectively no influence on its image in neighboring cells. This is possible because the force decays rapidly with distance. If this condition were not assured then the number of pair-wise interactions that must be considered would increase due to the



**Figure 1: Periodic Boundary Conditions in Two Dimensions**

replication of the cells. The typical approach is to assume a cut-off distance beyond which two particles are no longer considered to interact. The volume under consideration is then sized to be at least twice this length. This is known as the *minimum image criterion* [3].

The LJ potential gives the force on each particle but this information alone is not sufficient for an MD simulation. It is also necessary to determine how the particle's position and velocity are affected by this force. The first basic observation is that this force results in an acceleration given by  $F = ma$ , Newton's Second Law. Time integration will then be performed to update the position and velocity. Again, a variety of techniques are possible but this implementation will use the most widely used technique, the Verlet algorithm [4]. Within the Verlet algorithm, there are also various possibilities and the technique selected for this implementation will be the Velocity Verlet Update rule, hereafter, referred to as simply Verlet Update (VU). The following equations then are used for performing the position and velocity updates.

$$v(t) = v(t - \frac{\delta t}{2}) + \frac{\delta t}{2} \times a(t) \quad (4)$$

$$r(t + \delta t) = r(t) + \delta t \times v(t) + \frac{\delta t^2}{2} \times a(t) \quad (5)$$

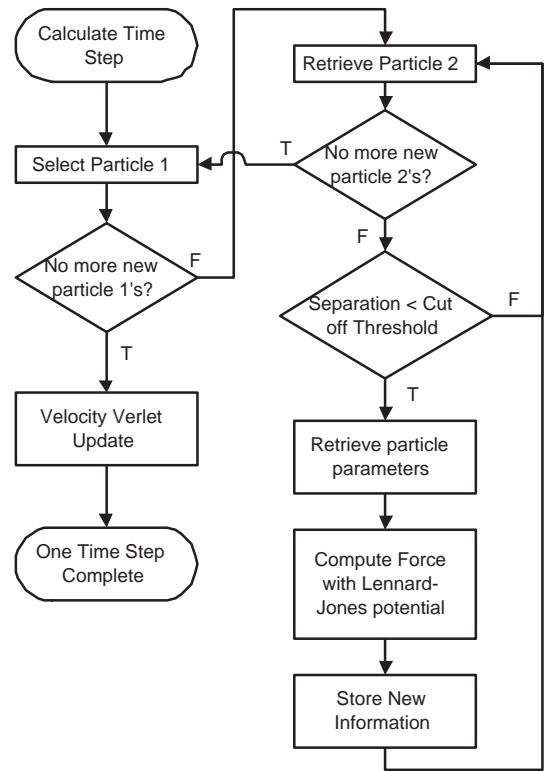
$$v(t + \frac{\delta t}{2}) = v(t) + \frac{\delta t}{2} \times a(t) \quad (6)$$

where  $\delta t$  is the time step. Using the basic steps described above the general procedure for performing a MD simulation is shown in Figure 2. Clearly the force computation presents a significant challenge as that portion of the algorithm is  $O(n^2)$ . The VU need only be performed once per particle and, thus, does not result in a significant bottleneck. Once a time step is complete, the process can continue until the desired length of time has been simulated.

## 2.2 Prior MD Work

A great deal of prior research has been directed at various aspects of MD simulations. Most implementations have focused on improving software algorithms for dealing with the  $O(n^2)$  computations, although there have been some implementations of application-specific hardware.

An Application Specific Integrated Circuit (ASIC) known as MODEL [5] was developed to calculate both the LJ potential (representing Van der Waal's forces) and the Coulombic force. Prior research had found that the computation



**Figure 2: MD Simulation Algorithm**

of these two forces required greater than 99% of the CPU time in software-based simulations. Only these non-bonded forces were considered and their magnitude was determined through table lookup. The system was designed to be highly parallelized. Using 76 MODEL chips, the MD simulation was approximately 50 times faster than the equivalent software implementation on a 200 MHz Sun Ultra 2.

Before a hardware implementation is possible, the necessary precision must be determined. While software implementations typically rely on the standard IEEE double-precision floating-point arithmetic, in hardware such precision is not always necessary. As well, to minimize hardware requirements it is desirable to minimize the required precision. An experimental approach to determine the necessary precision [6] found varying amounts of precision were needed for different stages of computation. The position vector was found to need a mantissa of 25 bits while the pairwise force computation required a 29 bit mantissa. The accumulator, which determines the net force on a particle required 48 bits in the mantissa. To determine these values, a one-dimensional system was simulated. The stability of the simulation was then observed for varying bit widths and the minimum for a stable simulation in terms of energy was considered to be the minimum required precision. The exponent length used was not reported in [6]. These results served as the basis for the work on the MODEL chip [5]. All the work was performed using floating-point number representations.

Another hardware implementation involved the design of a chip known as the MD-GRAPe [7, 8]. This work is interesting in that it did not use floating point arithmetic throughout as was done in MODEL [5]. Instead, position is

stored as a 40-bit fixed-point number and the force is accumulated onto an 80-bit fixed-point number. The switch to fixed-point reduces the hardware requirements substantially. However, floating-point was still used for the force computation. Again a lookup table implementation was used for the calculation of the force. The speed of the system though was reportedly somewhat slower than software implementations on supercomputers. Only the  $O(n^2)$  operations were off-loaded to the hardware in this system and  $O(n)$  operations were performed on a host computer. As a result until the number of particles becomes large the communication speed with the host limits the performance of the system.

### 2.3 TM3 Background

The TM3 [9] is an FPGA platform that contains multiple interconnected FPGAs. It consists of four Virtex-E 2000E devices connected to each other via 98-bit bidirectional buses. In addition, each FPGA is connected to a dedicated  $256k \times 64$  bit external SRAM, an I/O connector and a nibble bus which allows communication with the host computer for download and control functions.

## 3. ARCHITECTURAL DESIGN

This section provides an overview of the hardware design, and then provides a detailed description of the logic that makes up the components of the hardware system. The software infrastructure which handles both automatic customization of the simulation hardware and the interfacing with the hardware system is described in Section 4.

### 3.1 Top Level Description

A top-level block diagram for the MD system is shown in Figure 3. It is composed of six different blocks: the Pair Generator (PG), the Lennard-Jones Force Calculator (LJFC), the Acceleration Update (AU) block, the Verlet Update (VU) block, three memory controllers, and the System Controller (SC).

For every timestep in the simulation the PG examines a pair of particles in the system and determines the distance between the pair. This information is sent to the LJFC, which computes the force between the pair of particles given the distance between them. The AU accumulates the forces received from the LJFC for each particle. Once all the incremental forces have been received from the LJFC for a particular particle, the AU sends the total acceleration for that particle to the VU. Finally, the VU uses the total acceleration obtained from AU to update the position and velocity of each particle.

These steps are repeated until all pairs of particles have been examined, and then the whole process is restarted for the next timestep.

### 3.2 System Controller

The System Controller (SC) serves to coordinate the activities of the different blocks during the simulation of each timestep, and to signal the start of a new timestep. The SC also serves as an interface to the software modules running on a Sun Workstation.

### 3.3 Memory Controllers

Two types of memory controllers are used in the MD simulation system. The first type, named the Memory Controller (MC) block, is used in the primary FPGA and con-

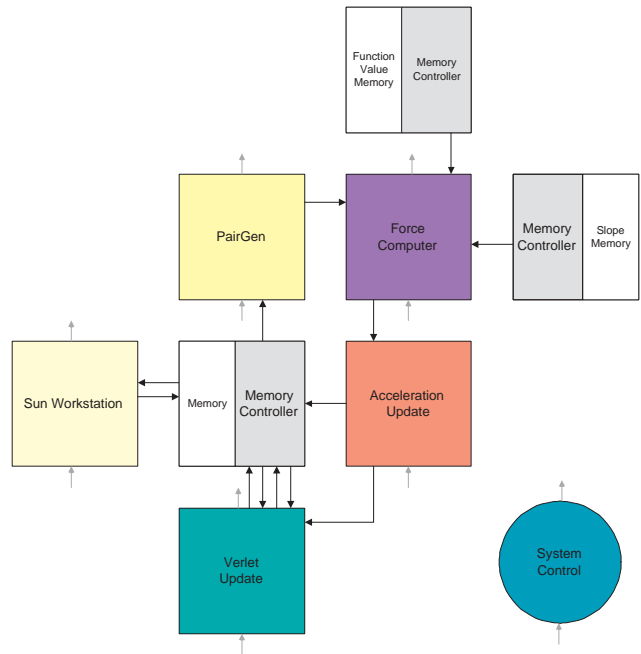


Figure 3: Top Level Block Diagram of MD System

trols access to the Static Random Access Memory (SRAM) bank that contains the position array (PA), the velocity array (VA) and the acceleration array (AA). The second type of memory controller, named the Lennard-Jones Memory Controller (LJMC) block, is instantiated in two other FPGAs to access two different lookup tables in other SRAM banks: one that stores the function values and one that stores the slopes of the LJ formula.

There are three blocks in the system that connect to the MC block to access the position, velocity, and acceleration arrays: PG, VU and AU. The PG block reads from the position array, the VU block reads and writes to both the position and velocity arrays, and the AU block writes to the acceleration array. The acceleration array is only read by the Sun Workstation. Since the position array is read and written to multiple times in each timestep there is the possibility that data from one timestep is overwritten before all the reads are complete. To eliminate this possibility, two position arrays are used. PG reads from one position array, and VU writes into the other array. After the completion of a every timestep, the MC will switch which position array is used for reading and writing.

Since the SRAM banks on the TM3 are single-ported, only one memory access can be performed at a time; thus, the MC block is designed to arbitrate between the MD simulation blocks and the external memory. It receives requests from the PG, AU and VU blocks, serves them based on a priority scheme, and then sends back the acknowledgment along with the output data (if the request was a 'read') to the corresponding block.

Each system-level read or write is in fact a set of three reads or writes: one for each X, Y and Z directions. Currently a 7-cycle system-level read and 6-cycle system-level write scheme is implemented. Since the memory access is the bottleneck in the current system, any improvement in memory access speed directly improves the overall system

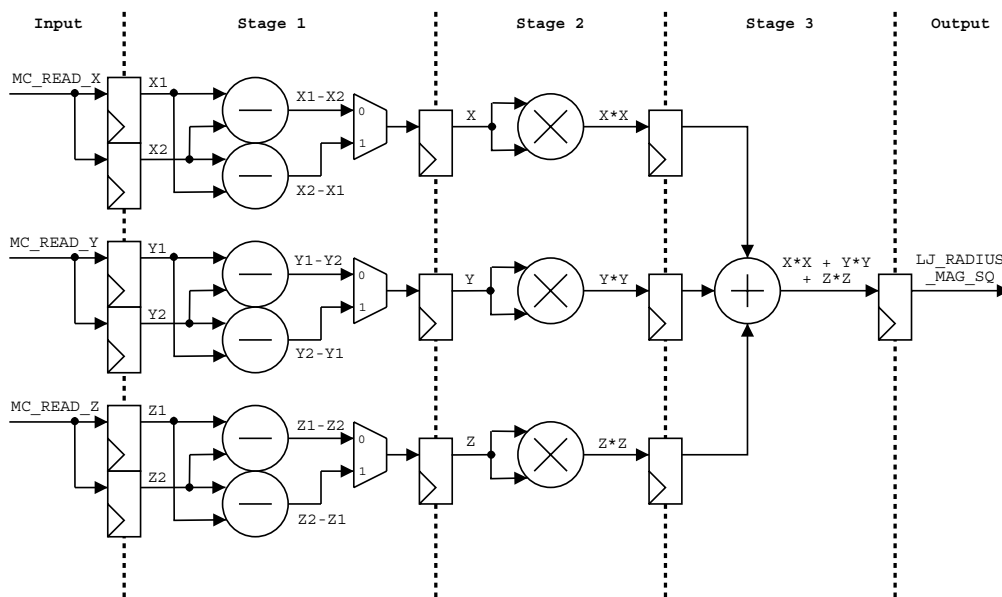


Figure 4: PG Datapath

performance.

The LJMC block is similar to the MC block described in the previous paragraphs. However, it is simpler than the MC block because it only receives read requests from the LJFC block. Therefore, there is no circuitry needed to arbitrate between different memory requests. Additionally, only one read operation is needed for each request as opposed to three read or write operations in the MC block.

### 3.4 Pair Generator

The Pair Generator (PG) block calculates the distance between all pairs of particles in the MD system. Its datapath is shown in Figure 4. The input signals on the left of the figure receive particle positions from the MC block. The data passes through three pipeline stages before reaching the output stage that connects to the LJFC block.

The first stage of the PG block calculates the distance between two particles in each dimension. Because of the periodic boundary conditions shown in Figure 1, the distance calculated must be the shortest between any mirror image of each particle. Therefore, a simple subtraction is not enough. Two subtractions are performed that rely on the natural roll-over of modular arithmetic to achieve the mirrored effect. The smaller of the two subtractions is chosen in each dimension. Not shown is that the direction of the force with respect to the first particle is determined by which subtraction was chosen.

The second stage of the PG block squares the X, Y and Z components of the distance between particles. Finally, the third pipeline stage sums the squares to get the square of the radius between the two particles and sends it to the LJFC block.

The PG block also contains a state machine that controls the fetching of position information from the MC block. The state machine begins by loading the position of the first particle from the MC block. Then it loads the position of the second particle to calculate the distance between the first two particles. Next, it loads the third particle to compare

with the first and continues until it reaches the last particle. Then it replaces the first particle with the second particle and begins comparing the second particle to all other particles starting with the third. The state machine completes after the distance between every pair of has been computed.

### 3.5 Lennard-Jones Force Calculator

The implementation of the Lennard-Jones Force Calculator (LJFC) uses the higher order bits of the radius squared,  $r^2$ , to lookup the value and slope of the LJ potential function. It then uses the remaining bits of  $r^2$  along with the value and slope to interpolate and obtain a more accurate acceleration value.

The lower order bits of the  $r^2$  value (PG\_RADIUS\_MAG\_SQ) in Figure 5) are registered in a residual register. These bits will be used later in the pipeline, along with the slope of the LJ potential, to interpolate and obtain a more accurate acceleration.

A number of the middle order bits are used as the address to the lookup table for both the LJ function value and the average slope of the LJ function at that point. The LJ function changes rapidly near distances of 0, first decreasing, reaching a global minimum and then slowly increasing. As the distance between particles becomes large the change in the force is very little and is near zero.

Due to this characteristic, it is more important to store more LJ function values in the lower range of the function (small values of  $r^2$ ). Given that only 19 bits could be used to look up a value from memory, if the high-order bits of the  $r^2$  were used to look up the LJ function value, then the function points stored in the lookup table would be spaced far apart, and the characteristic of the LJ function in between those points could be lost. By using the middle-order bits, the function is sampled more often when it is rapidly changing. However, by using the middle-order bits we are limiting the maximum distance that can be looked up, but this is not a problem since the LJ function is near 0 and changing very slowly at such distances.

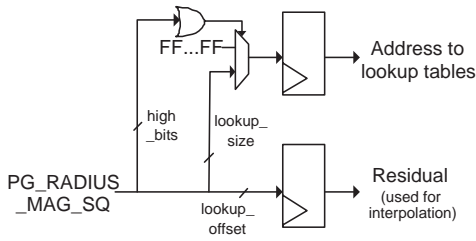


Figure 5: LJFC Lookup Address Logic

Figure 5 shows how the bits of the  $r^2$  value are used to lookup the LJ function. The middle-order bits are the `lookup_size` bits that go into the lower input of the multiplexer. The multiplexer is included to select the largest address possible (i.e. FF...FF) when any of the higher order bits are a logical '1' because the distance between the particles is larger than what can be represented by the middle-order bits, and thus the largest address should be used instead. Thus the higher order bits are ORed and used as the select signal into the multiplexer.

After the slope and force values are obtained from the lookup tables the residual  $r^2$  value is multiplied by the slope and stored in an increment register. Then this increment value is added to the LJ function value obtained from the lookup table. This value is not a true force or acceleration but a pseudo-acceleration; in fact the value includes a division by  $r$  and a multiplication by  $dt$  so that these two operations do not have to occur in hardware downstream in the pipeline.

The final operation is a multiplication of the pseudo-acceleration by the radius vector components to obtain the component pseudo-accelerations.

### 3.6 Acceleration Update

The role of the Acceleration Update (AU) block is to receive the incremental pseudo-accelerations from the LJFC and calculate the total acceleration for a particle. The AU functions by storing the sum of the incremental forces in a register. When all the incremental forces for a particle have been summed up, the AU stores the acceleration in memory and sends the result to the VU where the new position and velocity will be calculated as will be described in Section 3.7.

The AU needs to know when it has accumulated all of the incremental forces for a particle. Thus, when a new particle ID has been received from the LJFC, the accumulated value is sent to the VU and memory, and the accumulator is reset.

### 3.7 Verlet Update

The Verlet Update (VU) module is responsible for updating the position and velocity of each of the particles with the Velocity Verlet Update rule. In conventional software implementations this is typically performed in two different steps. In the first step the position is updated a full time step and velocity is updated to an intermediate half timestep value. The second step finishes the update by adjusting the velocity to the next timestep value. In hardware, to implement these two steps as is done in software, it would be necessary to have two Verlet modules with each performing the necessary updates. This is clearly inefficient so, instead, a single Verlet update is performed that combines both steps; however, as a result of using a single step, the velocity stored by

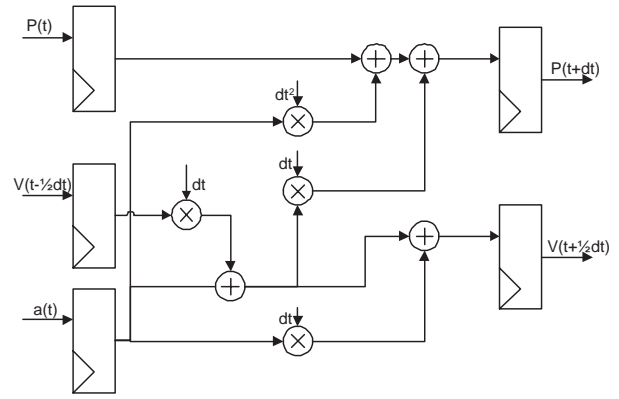


Figure 6: VU Block Diagram

the hardware is not the velocity at a given time step. In its place, the velocity at a half time step ahead of the current time is stored i.e. if we currently are at time  $t$  then the position will be  $r(t)$  but velocity will be  $v(t + 0.5\delta t)$ . This design decision will be invisible to the user as the interface software will adjust the velocity back to the appropriate time. Clearly such adjustment could be done in hardware; however, the system was designed with the goal of simulating many time steps at once. When multiple time steps are simulated, the amount of time spent back-stepping the velocity in software is trivial and, thus, the added complexity of allowing the VU module to perform partial time steps was not warranted.

The VU module requires numerous multiplications. In particular acceleration is always multiplied by a factor of time. To simplify the hardware, this multiplication by time will be included in the acceleration computed by the LJFC. Including such a factor is trivial as it only requires that the lookup tables be multiplied by a factor of time, which can be done in software prior to simulation and thus has no effect on simulation speed. As a result, the pseudo-acceleration given to the VU does not represent true acceleration. Again the software subsystem can adjust the contents when acceleration must be read on the host computer.

As discussed in Section 2.1, the VU operation is  $O(n)$  and, compared to the  $O(n^2)$  portions of the system, this block will not critically limit the performance of the entire system. The top-level diagram depicting the essential elements of the VU module is shown in Figure 6.

### 3.8 Precision and Scaling Factors

While the MD system presented here may seem to implement the required functionality, the results obtained depend heavily upon the precision and scaling factors used in the system. With too low a precision, the results from the MD simulation are inaccurate, and with too high a precision, valuable hardware resources are being wasted and the system will be slowed down. In this section the methodology behind choosing the scale factors and precision is outlined, and the final values chosen for the precision and scaling factors for different fields within the design summarized.

The different quantities used within the MD system all have different precision and scaling factors. By precision, it is meant the number of bits used to represent the field, and by the scaling factor it is meant the weight of the least significant bit of the field. For example if a particular field

Table 1: Scaling Factors and Precision of Various Fields

| Field                              | Used in Block | Scaling Factor | Precision |
|------------------------------------|---------------|----------------|-----------|
| $r$                                | PG            | $2^{-64}$      | 38        |
| $r^2$                              | PG LJFC       | $2^{-128}$     | 76        |
| slope                              | LJFC          | $2^{77}$       | 36        |
| function value                     | LJFC          | $2^{16}$       | 47        |
| residual $\times$ slope            | LJFC          | $2^{15}$       | 48        |
| pseudo-acceleration                | LJFC AU       | $2^{-14}$      | 50        |
| velocity                           | VU            | $2^{-15}$      | 51        |
| velocity after<br>step 2 of Verlet | VU            | $2^{-15}$      | 51        |
| acceleration $\times dt$           | VU            | $2^{-64}$      | 37        |
| velocity $\times dt$               | VU            | $2^{-64}$      | 37        |
| representation of $t$              | VU            | $2^{-70}$      | 22        |

has 8 bits of precision, a scaling factor of  $2^{-2}$  and holds the bit pattern, 00010111, then the field would represent the number  $23 \times 2^{-2}$  or equivalently 5.75.

Prior work by Amisaki in [6] is not directly applicable as it relied on floating point implementations while this system is entirely fixed point. However, a similar but simpler approach was taken to determine the scale factors needed in the system. Computations for a two-particle MD system were performed in a spreadsheet. The computations were completed assuming the architecture presented above and the values of all fields (i.e. registers in the design) were displayed in the spreadsheet.

Calculations were made with the particles at varying distances ranging from extremely close together to very far apart. This is a reasonable approach since the behavior of the force function is known and thus samples could be taken at the points of interest. The distance range studied included all reasonable regions where the force is of interest. For extremely distant particles the force becomes negligible and can be ignored. The repulsion between particles prevents them from moving within the minimum distance. After taking the absolute value of all the calculated values, the minimum and maximum values for every field were found.

The scaling factor of the field was determined by taking the  $\log_2$  of the minimum value of the field and the precision of the field was determined by taking the difference of the  $\log_2$  of the maximum and minimum value of the field.

For some fields, such as acceleration, where a value would be accumulated in the field, extra precision was added in the upper portions of the field. Furthermore, to be on the safe side, extra bits were added to the upper and lower portions of most fields, and the scaling factors adjusted. Table 1 shows the final scaling factors and precision used for various fields within the design.

In the MD design the inputs and outputs of various arithmetic operations do not have consistent scaling factors and precision. Therefore, generic adders and multipliers were designed in VHDL that take six generic inputs: the scaling factors and precisions of each of the two inputs, and the scaling factor and precision of the output. This adder and multiplier perform the required shifts, concatenations, and truncations on the inputs and outputs to create the desired operation.

## 4. SIMULATION RECONFIGURABILITY

The MD system is designed to make use of FPGA’s inherent reconfigurability. The design is not static and instead is generated based on the user’s simulation requirements. A software system was developed to simplify the process of performing MD simulations. By preparing a configuration file, the user is able to control the simulation environment setting such variables as precision, scaling factors and the number of particles to simulate. The software generates VHDL describing the specific design which is then synthesized. This ability to tailor the hardware to the required simulation is extremely powerful since the user’s simulation is never burdened by the hardware that would be needed for a generic simulation environment.

Finally the software tools also control the interface with the FPGA. Based on the user’s desired configuration the hardware memories are initialized with the correct lookup table values as well as the particle starting positions and velocities. The user can then monitor and control the hardware as the simulation progresses by downloading the memory contents. Key simulation results such as kinetic and potential energies are then generated by the software. Since these parameters are only required occasionally calculation in software is best as it significantly simplifies the hardware design.

## 5. RESULTS

The software and hardware system described above was then implemented on the TM3. Thorough testing confirmed that the system performed as specified. However, to ensure the usefulness of this work, two significant issues must be considered, the validity of the approach taken and the performance with respect to software implementations.

### 5.1 Validation

As described earlier most software implementations utilize IEEE double-precision floating-point throughout the calculations but to minimize the hardware requirements this system used fixed-point numbers whose widths were determined experimentally. To test that the approximations used are reasonable, simulation results were compared between a software implementation that is known to be good and the hardware implementation.

The software model used was an academic C-based MD simulator, the MD3DLJ, from the Institute of Computer Science at the Academy of Mining and Metallurgy in Cracow, Poland [10]. This code performs a simulation on a periodic boundary condition system using Lennard-Jones forces.

A number of minor modifications were made to this code to allow a comparison to the hardware system. The code was modified to use the exact constants and system parameters that were used in the hardware.

The first most basic comparison between hardware and software is a particle by particle comparison of positions after a single timestep. Given the numerical simplifications made in hardware, the two results will not be identical but the results should be reasonably close. This was found to be the case with the error on the order of 1% RMS.

The system was then tested over longer simulation periods. One such comparison is shown in Figure 7. A closed system is being modeled so it is expected that total energy remain constant and this clearly holds true for both the hardware and software implementations. However, the

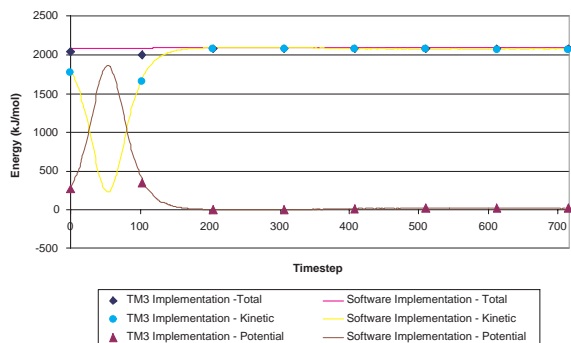


Figure 7: Comparison of Hardware and Software Models

Table 2: Comparison of Energy of TM3 vs Software Model

| Time Step      | Kinetic Energy (KJ/mol) | Potential Energy (KJ/mol) | Total Energy (KJ/mol) |
|----------------|-------------------------|---------------------------|-----------------------|
| Initial        | 1794                    | 293.4                     | 2087                  |
| Software - End | 2078                    | 14.40                     | 2093                  |
| Hardware - End | 2068                    | 13.52                     | 2081                  |

more interesting result is that both hardware and software experience similar changes in potential and kinetic energy. Figure 7 shows the TM3 energy values and results tracking the software results almost exactly. The similarity of the results can also be seen numerically in Table 2. Both simulators experienced the same jump in kinetic energy and the difference between the total energy was less than 1%.

## 5.2 Error Analysis

To check the original precision and scaling values determined in Section 3.8, a more detailed analysis of the step-by-step computations was done. A version of the software model was modified so that the fixed-point hardware could be emulated and run concurrently with the double-precision versions of the same calculations. Units of the hardware could be emulated in isolation meaning that full double-precision calculations could be done up to the entry of the hardware unit, converted to the fixed-point representation, and then computed in the emulation of the hardware unit. At the same time, the results were also computed in full-double precision for that unit.

The results showed that there was enough dynamic range at all steps, which is not surprising based on the original method of Section 3.8.

## 5.3 System Performance and Size

As discussed earlier, one of the major benefits of the MD system, as it has been designed on an FPGA, is that the entire system that can be reconfigured at compile time to change various simulator attributes, such as the precision, and the number of particles. Thus, timing numbers are dependent on the options selected. For the 8192 particle system validated in the previous section a maximum clock frequency of 26.063 MHz was achieved. This gives a per time step performance of 37.0 seconds. The software imple-

Table 3: Comparison of MD Simulator Performance

| MD Simulator                            | Timestep (s) | Rel. Speedup to Software |
|---|--------------|--------------------------|
| TM3 (26MHz)                             | 37           | 0.29                     |
| Software (2.4 GHz P4)                   | 10.8         | 1.0                      |
| Better Memory System                    | 2.1          | 5.1                      |
| Faster FPGA (100 MHz) and Better Memory | 0.51         | 21                       |

mentation running on a 2.4 GHz Pentium 4 is able to simulate a time step in 10.8 seconds. Two factors contributed to the slow FPGA implementation: limited memory bandwidth and FPGA speed. Table 3 shows the performance comparisons.

The resulting circuit uses about 18906 LUTs and could fit in one FPGA with a few chips of external memory for the LJFC lookup table.

## 6. IMPROVING PERFORMANCE

Given the performance reported in Section 5.3 the system seems to have no advantage over a software system, but the reason for the long 37 second timestep on the TM3 has to do with the memory architecture of the TM3 and not with the design. This memory bottleneck on the relatively old TM3 system was expected from the beginning of the design process, but the point of creating the MD system was to show if given a reasonably designed memory system, can an FPGA system provide any benefits compared to software-based supercomputers.

In this section it will be explained how the memory architecture of the TM3 limited the MD project and the rate that could have been obtained given a better memory architecture. We also discuss improving the clock speed and then show that the MD design is scalable and is parallelizable to larger systems.

### 6.1 Speedup with Better Memory System

The MD design needs memory to store positions, velocities, and accelerations. For an 8196 particle system, these memory requirements are a modest 0.69MB. An additional 2MB is needed for the function and slope calculation in the LJFC; however, this amount does not vary with system size.

Effective organization of the memory is important because of the many blocks that access the system memory. The velocity array is only used by the VU and the acceleration array is only used by the AU. That leaves the position array, which is read by the PG and is read and written to by the VU. To alleviate the contention on the position array, the position array in this design was divided into two arrays; one for reading by the PG and the VU and one for writing by the VU. At the end of the timestep, these two arrays are switched.

Since there is only one external SRAM array per FPGA on the TM3, and the BlockRAMs on the FPGAs on the TM3 are too small to hold the data, the position, acceleration, and velocity arrays had to be placed on the same SRAM. This caused a considerable slow-down in the performance of the MD system. The slow-down due to the TM3 memory architecture can be broken down into the following components:

1. Factor of 3 slowdown due to the handshaking needed

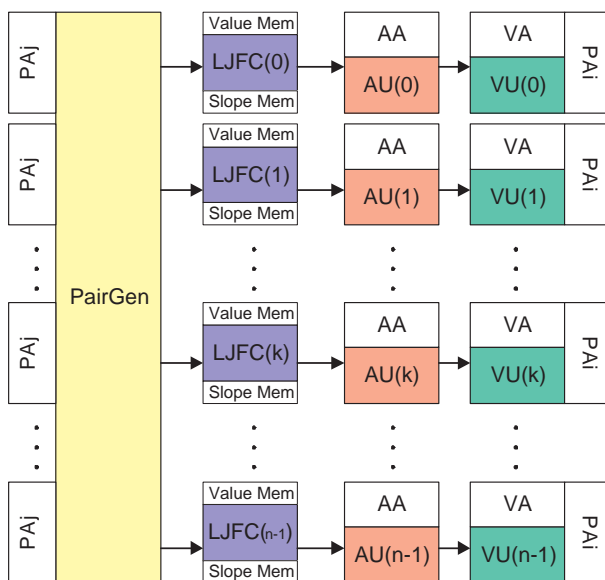


Figure 8: Scalable Parallelized MD System

- for arbitrating accesses to the single memory array.
- Factor of 3 slowdown because the width of the data bus was too small so the x, y, and z components of position/velocity/acceleration were accessed serially instead of in parallel.
- Factor of 2 slowdown due to wait states required by the memory system (a TM3 issue) between each access.

If a better memory system was available, the current MD system could be sped up by a factor of 18, thus obtaining a 2.1 second timestep, which would be about 5 times faster than the software system as shown in Table 3. Notice that no extra memory has been added, but the memory has just been reorganized.

## 6.2 Clock Speed

By porting the system to a more modern FPGA a higher clock speed should be easily achievable. We have also not explored all possible design improvements that could allow for faster clock speeds. With design improvements and updated FPGAs a factor of 4 increase in speed to 100MHz is a conservative estimate of what can be achieved. Other experience has shown that with reasonable pipelining 100 MHz circuits can be achieved in Virtex II and Stratix devices.

The last entry in Table 3 shows what could be achieved with a better memory system and the higher clock rate. This system would be about 20 times faster than the software simulator.

## 6.3 A Parallel Architecture

To add parallelism, the technique is the same as is done in multiprocessor systems. Multiple MD engines, like the one implemented in this project, are put in parallel. Figure 8 shows a possible implementation. The LJFC, AU and VU blocks are exactly the same as in the current MD system and can fit in one medium-size FPGA. Three additional 512K×32 memory chips would be needed for the LJFC lookup table for each pipeline and the remaining memory would be internal to the FPGAs. The acceleration array (AA), velocity array (VA) and position array (PA) are distributed throughout the system, and do not need any ad-

ditional memory compared to the serial architecture. Note that PA<sub>i</sub> and PA<sub>j</sub> are the two sets of position arrays that need to be swapped after the completion of each timestep as explained in Section 3.3.

One other property that was observed in this project and used in other large software-based systems is that by using cut-off, it is possible to significantly reduce the number of pairs that need to be considered. Through software simulation, it was noticed that in a system with a large number of particles, 98.9% of particle pair comparisons result in pairs beyond the cut-off distance so the force does not need to be computed. This means that a more sophisticated PG block could be developed to take advantage of this property so that not all possible pairs need to be examined.

## 7. FPGAS FOR SUPERCOMPUTING

The implementation of our MD system has provided a good basis for estimating the performance benefits of ASPs based on FPGAs. In this section we also do an exercise to examine the benefits in terms of cost, power, and size.

### 7.1 Cost, Power, Size Benefits

We have shown that FPGAs can be used to implement an interesting application that has typically been targeted at high-performance computing systems and that significant performance gain can be achieved. In this particular case, we have shown that with modest hardware design constraints (a 100 MHz clock is easily achievable in pipelined systems) about a 20x performance improvement can be achieved over a modern microprocessor system. With a more ambitious design, a higher clock rate could be achieved, providing even greater performance. However, let us assume a conservative 10x improvement to allow for the fact that even faster microprocessor systems are available, and that there are better compilers available so the performance advantage we have measured is somewhat reduced.

Besides performance, other important system-level issues are cost, power and physical size (footprint). We construct a plausible system and do some approximations to investigate the benefits of an FPGA-based system.

A high-performance microprocessor-based motherboard will contain many components that are not required for an FPGA-based application. The main simplification would be in the memory system where caches would not be required, and for MD, large multi-gigabyte memories are also not needed. If required, small, fast memories can be implemented inside the FPGAs. In the same space as a CPU motherboard, it is possible that at least 4 FPGAs could be placed. Assuming that the FPGAs cost about \$200 each (we do not need the largest, most expensive devices), then the FPGA cost for 4 FPGAs would be about \$800, so the board would likely cost less than \$1500, roughly the same as a good CPU motherboard. However, for highly parallelizable applications like MD, this board would provide about 40x in performance for the same cost (performance/cost ratio).

If each FPGA and SRAM combination runs at about 10W of power (calculated using a full largest Altera Stratix FPGA [11] at 6.2 Watts and 6MB of SRAM running at 2.7W [12]), then the 4-FPGA board would run at about 40W, which is significantly less than a CPU motherboard composed of a Pentium®-4 running at 82 Watts [13] and 1Gig of DDR RAM running at 24 watts [14]. This is about a 100x improvement in the performance/power ratio assum-

ing a 10X performance improvement for the FPGA implementation.

For physical size, our assumption started with the same size of board for both the CPU and the FPGA-based system. Given that the FPGA board will have about 40x performance improvement, then the performance/size ratio shows a 40x improvement in favor of the FPGA-based system.

The above simple rough calculation demonstrates that FPGA-based supercomputing systems can show about 40x to 100x improvement over a microprocessor CPU-based system when considering performance relative to cost, power and size. The computing power of a supercomputer needing a room the size of a gymnasium could fit into the space of a large file cabinet.

## 8. CONCLUSION

This paper documents the successful implementation of an MD simulator on the TM3. The system is able to simulate an 8192 particle system at a rate of 37 seconds/timestep while running at 26 MHz, and we show that with minor modifications in the memory organization the same system would complete a timestep every 2.1 seconds.

The system uses fixed-point representations for all values within the system thus reducing the complexity and area needed compared to floating-point systems. We show that the results of the fixed-point hardware implementation closely track those of higher precision software implementations with an error of less than 1% between consecutive time steps. In longer simulations, the error in potential energy remained below 1% while kinetic energy differences between hardware and software were less than 5%. Later analysis has shown how these inaccuracies can be improved.

The designed system is versatile and configurable due to nature of the system being implemented on an FPGA. In the past, ASICs were necessary but thanks to the advanced capabilities of modern FPGAs the entire system can be implemented on essentially a single FPGA. The switch to FPGA development offers numerous advantages. With the adjustment of a few parameters in a single file and a matter of hours, an entirely new hardware system can be generated that can cope with a different amount of particles, with different particle types, and most importantly with a varying amount of precision; for example, if a quick simulation is needed on limited hardware an MD system can be synthesized with relatively low precision, or if large FPGAs are available, and high precision is needed, another MD system can be synthesized.

The successful implementation of this system shows that FPGAs are viable as processing elements in high-performance computing applications providing significant cost, power, and area savings. As well, the addition of a software layer that makes it easy to configure various aspects of the hardware before the circuits are created makes the overall MD system much more flexible and useful. This is an excellent example of how to use FPGAs for reconfigurable processing.

Future work will be done to develop the multi-FPGA system and include the Coulombic forces.

## Acknowledgments

The authors would like to thank Paul Leventis and Terry Borer for their help and support in creating the MD System.

## 9. REFERENCES

- [1] Kai Hwang, editor. *Supercomputers: Design and Applications*. IEEE Computer Society, 1984.
- [2] Norman Margolus. An Embedded DRAM Architecture for Large-Scale Spatial-Lattice Computations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 149–160. IEEE/ACM, 2000.
- [3] Furio Ercolessi. A molecular dynamics primer. <http://www.fisica.uniud.it/ercolessi/md/md.pdf>, Jun 1997.
- [4] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, 1987.
- [5] Shinjiro Toyoda, Hiroh Miyagawa, Kunihiro Kitamura, Takashi Amisaki, Eiri Hasimoto, Hitoshi Ikeda, Akihiro Kusumi, and Nobuaki Miyakawa. Development of md engine: High-speed acceleration with parallel processor design for molecular dynamics simulations. *Journal of Computational Chemistry*, 20(2):185–199, 1999.
- [6] Takashi Amisaki, Takaji Fujiwara, Akihiro Kusumi, Hiroo Miyagawa, and Kunihiro Kitamura. Error evaluation in the design of a special-purpose processor that calculates nonbonded forces in molecular dynamics simulations. *Journal of Computational Chemistry*, 16(9):1120–1130, 1995.
- [7] Toshiyuki Fukushige, Makoto Taiji, Junichiro Makino, Toshikazu Ebisuzaki, and Daiichiro Sugimoto. A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: Md-grape. *The Astrophysical Journal*, 468:51–61, 1996.
- [8] Yuto Komeiji, Masami Uebayasi, Ryo Takata, Akihiro Shimizu, Keiji Itsukashi, and Makoto Taiji. Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer. *Journal of Computational Chemistry*, 18(12):1546–1563, 1997.
- [9] TM-3 documentation. <http://www.eecg.utoronto.ca/~tm3/>.
- [10] MD3DLJ A PC/workstation C - language Program for L-J Molecular Dynamics in 3 - Dimensions. Technical report, Institute of Computer Science Academy of Mining and Metallurgy, Cracow, Poland, 1989. <ftp://ftp.dl.ac.uk/ccp5/MD3DLJ.C>.
- [11] Stratix power calculator. [http://www.altera.com/products/devices/stratix/utilities/power\\_calculator/stx-power\\_calculator\\_download.html](http://www.altera.com/products/devices/stratix/utilities/power_calculator/stx-power_calculator_download.html).
- [12] Cypress CY7C1062AV33 512×32 static RAM datasheet. <http://www.cypress.com/cfuploads/img/products/CY7C1062AV33.pdf>.
- [13] Intel®Pentium®4 Processor with 512-KB L2 Cache on 0.13 Micron Process Datasheet. <http://developer.intel.com/design/pentium4/datashts/298643.htm>.
- [14] Samsung 512Mb B-die DDR SDRAM Specification. [http://www.samsung.com/Products/Semiconductor/DRAM/DDRSDRAM/DDRSDRAMcomponent/512Mbit/K4H511638B/DDR\\_512Mb\\_B-die\\_TSOP2\\_x4x8x16\\_rev1.1.pdf](http://www.samsung.com/Products/Semiconductor/DRAM/DDRSDRAM/DDRSDRAMcomponent/512Mbit/K4H511638B/DDR_512Mb_B-die_TSOP2_x4x8x16_rev1.1.pdf).