# A Custom FPGA Processor for Physical Model Ordinary Differential Equation Solving

Chen Huang, Frank Vahid, and Tony Givargis

***Abstract*—Models of physical systems, such as of human physiology or of chemical reactions, are typically comprised of numerous ordinary differential equations (ODEs). Today's designers commonly consider simulating physical models utilizing field-programmable gate arrays (FPGAs). This letter introduces a resource efficient custom processor—the differential equation processing element, or DEPE—specifically designed for efficient solution of ODEs on FPGAs, and also introduces its accompanying compilation tools. We show that a single DEPE on a Xilinx Virtex6 130T FPGA executes several physiological models faster than real-time while requiring only a few hundred FPGA lookup tables (LUTs). Experiments with a commercial high-level synthesis(HLS) tool show that while a single DEPE is 5–50× slower than HLS circuits, DEPE is 10–200× smaller. We show that a single DEPE is only 10× slower than a relatively massive and costly 3 GHz Pentium 4 desktop processor for ODE solving, and its speed is also competitive with a 700 Mhz TI digital signal processor and an 450 Mhz ARM9 processor. DEPE is 4×–17× faster than a Xilinx MicroBlaze soft-core processor and 3×–6× smaller. DEPE thus represents an excellent processing element for use by itself for small physical models, and in future parallel networks for larger models.**

***Index Terms*—Custom processor, field-programmable gate array (FPGA), ordinary differential equation (ODE) solving, physical model simulation.**

## I. Introduction

THE capture and execution of physical models have gained extensive research attention in past decades. A physical model is a mathematical representation of a physical phenomenon. Executing a model may assist with understanding a physical process such as a human heartbeat [10] or with testing control algorithms in a cyber-physical system. Models commonly execute several orders of magnitude slower than real-time. Faster execution is clearly preferred, and in fact real-time execution is mandatory for much cyber-physical system testing. Real-time execution is possible today for more models due not only to faster processors, but also to larger capacities of field-programmable gate arrays (FPGAs).

Manuscript received April 08, 2011 ; accepted June 16, 2011. Date of publication September 29, 2011; date of current version December 21, 2011. This manuscript was recommended for publication by J. Hoe. This work was supported in part by the National Science Foundation (CNS1016792) and by the Semiconductor Research Corporation (GRC 2143.001).

C. Huang and F. Vahid are with the Department of Computer Science and Engineering, University of California Riverside, Riverside, CA 92507 USA (e-mail: chuang@cs.ucr.edu; vahid@cs.ucr.edu).

T. Givargis is with the Department of Computer Science and the Center for Embedded Computer Systems, University of California Irvine, Irvine, CA 92697 USA (e-mail: givargis@uci.edu).

Digital Object Identifier 10.1109/LES.2011.2170152

FPGAs implement circuits, wherein thousands of connected components execute in parallel. Circuits represent an excellent match for executing physical models. A physical model typically consists of thousands of ordinary differential equations (ODEs), each ODE representing part of the physical space, each part connected with neighboring parts, all parts executing in parallel. Execution consists of solving the ODEs using known iterative techniques. Thus, ODE solvers can be mapped to circuit components, each connected to neighboring components, thus avoiding the memory or input/output bottlenecks common in FPGA applications, and each executing in parallel.

Many previous case studies of using FPGAs to speedup physical system simulation have been conducted. For example, Yoshimi [17] obtained 100X speedups of a fine-grained biological simulation compared to a processor. Pimentel [11] used FPGAs to simulate a heart-lung system in real-time for the purpose of testing medical devices. Those case studies mostly used manual circuit design and optimization, which required significant human effort for design and test.

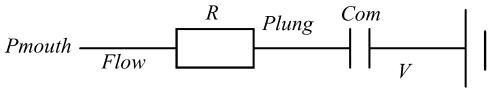
High-level synthesis (HLS) [2], [5], [14] compiles equations into circuits, but the large numbers of equations in physical models quickly overloads HLS. Instead, the equations can be partitioned among a network of programmable processors. The size of such processors will be a key determining factor of the execution speed of the model, because the processor size determines the number of processors that can fit on an FPGA.

This letter introduces a resource efficient custom programmable processor designed for fast ODE solving. Using a processor rather than custom circuit may also provide future benefits involving instrumenting the processor code to support profiling, debugging, updating, and more. We overview of physical modeling, and introduce our "DEPE" processor's architecture and describe the processor's supporting compiler. We compare with a commercial HLS tool and with several programmable processors including a soft-core FPGA processor and an application-specific instruction-set. DEPE is useful by itself for small physical models, but more importantly provides a good basis for future parallel processor networks on FPGAs for larger models.

## II. Physical Modeling and ODE Solving

Fig. 1 provides a basic model of a human lung [8] known as a resistance/compliance, or RC, model. The amount of air flow into the lung ($Flow$) equals the air pressure at the mouth ($Pmouth$) minus the pressure at the lung ($Plung$), divided by the resistance ($R$) of the lung. Furthermore, $Plung$ equals the lung volume ($V$) divided by the lung compliance ($Com$). $Flow$ equals the derivative of lung volume with respect to time.

A more complex model may be obtained by replicating equations. For example, Weibel [15] modeled a human lung as com-




Flow = (Pmouth – Plung) / R
Plung = V / Com
ODE: d(V)/dt = Flow

Fig. 1. Simple "RC" lung model.

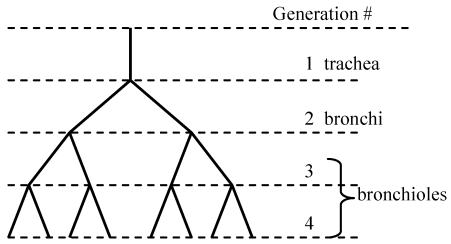



Fig. 2. Four generation Weibel lung.

prised of multiple generations (over 20) as in Fig. 2. Each branch in each generation is modeled using an ODE for Volume (similar to Fig. 1) and another ODE for Flow. More generations in the model yields higher accuracy but exponentially increases the number of ODEs. For simplicity, we refer to a system of ODEs of dimension $m$ as $m$ ODEs.

To determine the lung volume at a given time, the ODE can be solved using differential equation solution techniques. Instead, solutions are approximated using iterative solvers such as Euler [1] or Runge-Kutta [4]. Starting from time 0, iterative solvers move forward in time by a given *time step*, such as by 1 ms. At each time step, two major tasks are performed.

1) *Evaluate*: Estimate each variable's derivative value using the equation slope for current variable values, e.g., $d(V)/dt = (Pmouth - V/Com)/R$
2) *Update*: Approximate variable values for the next time step using the current values and the derivatives calculated above, e.g., $V = V + d(V)/dt * h$, where $h$ is the time step.

The Euler method performs the above tasks once per step, and has an error rate proportional to $h^2$ per step. Runge-Kutta (RK) methods give better accuracy. The RK4 method calculates the derivative of each variable four times—at the beginning, midpoint (twice), and end of the interval—per time step, with error proportional to $h^5$ per step (note $\mathrm{h} < 1$). An intermediate RK2 method gives an error rate of $h^3$ per step.

For this letter, we implemented three physical models. The first is a Weibel lung model [15]. We considered 2 generations (yielding 6 ODEs) and 4 (30 ODEs). The second model is a Lutchen lung [9], which models the gas exchange of the lung rather than the physical structure. It has a complex gas exchange model for the airway, consisting of a number of "cells" (segments in the airway for gas exchange). We considered 10 cells (yielding 10 ODEs) and 50 (50 ODEs). The third is a heart model aiming to represent heart failure [16], containing 31 ODEs. We thus had 5 physical models: Weibel 2/4 (denoted W2, W4), Lutchen 10/50 (L10, L50), and the heart model. We use small models in this letter, because the DEPE is designed for solving a simple physical model (<100 ODEs) or part of a complex model.

TABLE I
HLS PERFORMANCE AND RESOURCE USAGE

| Model | Perf (ms) | LUTs | DSPs | BRAMs |
|---|---|---|---|---|
| W2 (6 ODEs) | 2.7 | 2,110 | 41 | 0 |
| L10 (10 ODEs) | 2.3 | 2,602 | 151 | 0 |
| W4 (30 ODEs) | 2.9 | 9,233 | 215 | 0 |
| Heart (31 ODEs) | 3.3 | 9,775 | 124 | 0 |
| L50 (50 ODEs) | 2.4 | 62,945 | 360 | 0 |

*Target FPGA*: We targeted a Xilinx XC6VLX130T-3 FPGA, having 80 000 LUTs (lookup tables), 480 DSP units (built-in hardcore multipliers), and 264 BRAMs (built-in 32Kb hardcore block RAMs). We used Xilinx ISE 12.3 for synthesis. The work is not limited to a particular FPGA or synthesis tool.

*Method*: In this work, we used the RK4 solver with a 0.0001 s time step for all experiments.

## III. HIGH-LEVEL SYNTHESIS AND A MICROBLAZE SOFTCORE

We first investigated using high-level synthesis (HLS) to implement physical models on FPGAs. HLS converts equations into a circuit consisting of a controller and datapath. Modern HLS tools perform extensive algorithm parallelization (e.g., loop unrolling) and create heavily-pipelined datapaths.

Unfortunately, HLS tools typically do not generate resource efficient circuits for large sets of equations. To illustrate, we experimented with a commercial HLS tool.[1] We initially expanded the equations into the corresponding RK4 equations [see Fig. 4(b)] captured in C, but HLS generated circuits were huge. We instead just synthesized an ODE kernel for the Evaluate task of Section II, and manually wrote an HDL wrapper for the RK4 solver that utilized the HLS generated ODE kernel four times per step.

Table I shows HLS results. *Throughout this letter, performance numbers are in milliseconds (ms) and are normalized to 1000 ms of simulated time*, so 3 means the model executed 1 s of simulated time in just 3 ms (thus executing much faster than real time). Though the performance is much faster than real-time, HLS required tens of thousands of LUTs for the W4 and Heart models, and over 60K LUTs for the L50 model. It also used several hundred DSP units. Note that L50 model uses about 80% of LUTs in the target FPGA.

The input to HLS can be further modified to improve results. One could partition the equations while generating them, or use a partitioning algorithm to divide the generated equations. Rather than investigating this avenue further, which might yield some resource savings but not order-of-magnitude savings, we chose to investigate options involving a programmable processor for much larger savings.

Xilinx provides the MicroBlaze soft-core processor optimized for Xilinx FPGAs. For our target FPGA, the default MicroBlaze version uses 1445 LUTs, three DSP units, and eight BRAMs, and runs at 123 MHz. We create C code for RK4 solutions for our five models [as in Fig. 4(b)] and used

[1]Licensing restrictions prevent disclosing the tool name. The tool is used by dozens of industry firms and by the US military.

TABLE II
FPGA SOLUTIONS' PERFORMANCE (MS)

| Model | HLS | MicroBlaze | ASIP | DEPE |
|---|---|---|---|---|
| W2 (6 ODEs) | 2.7 | 185 | 12 | 11 |
| L10 (10 ODEs) | 2.3 | 98 | 20 | 19 |
| W4 (30 ODEs) | 2.9 | 430 | 71 | 66 |
| Heart (31 ODEs) | 3.3 | 383 | 94 | 87 |
| L50 (50 ODEs) | 2.4 | 506 | 117 | 108 |

Xilinx's compiler to compile to the MicroBlaze with the $-$O2 optimization. Performance results are shown in Table II.

The MicroBlaze is on average 100$\times$ slower than HLS, but faster than real-time for all five simple models, though W4, Heart, and L50's simulation time is getting close to 1 s.

## IV. DIFFERENTIAL EQUATION PROCESSING ELEMENT

### A. Initial ASIP Design

We developed an application-specific instruction-set processor (ASIP) [7] for differential equation solving for improved speed and smaller size versus a MicroBlaze. Only a few instructions are needed to support a differential equation solving application. Based on the application, we created an instruction set with just 12 instructions (load, load const, store, store_din, add, sub, mul_shift, and, or, xor, jpnz, jpz). Note that a store_din instruction stores data from an external input, and a mul_shift instruction assists with fixed-point multiplication. We created a standard pipelined controller to handle instruction fetch, decode, and execution.

ASIP performance for the five models is shown in Table II. The ASIP executes the models on average 5$\times$ faster than the MicroBlaze. Resource usage will be discussed shortly.

### B. NISC Processing Element—DEPE

We noted that an instruction set, even a small 12-instruction one, might not be necessary for the intended purpose. Eliminating the instruction set, and instead filling the instruction RAM directly with datapath control words, LUTs and increase clock frequency. This is the principle behind the no-instruction-set computer (NISC) [12].

The DEPE datapath shown in Fig. 3 has three input ports and one output port for communicating with external variables (or with other processors in future work), with those numbers being adjustable to the physical model. The datapath has a dual-port Data RAM serving as a register file. The Data RAM size and implementation (LUTs or BRAM) can be changed for the model. The default ALU has an integer adder/subtractor, and a DSP unit to perform integer multiplication; other components, such a divider, can be included if needed by the model.

Executing a model required use of just two different types of control words. A *compute* control word executes $dram[i]\ op\ dram[j] -> dram[k]$, i.e., read two Data RAM items, perform an ALU operation, and write the result in Data RAM. A *store* control word executes $din[i] -> dram[j]$, i.e., store one of the input ports into Data RAM. Branching instructions can be eliminated, because ODE solving process only involves sequential computations per time step. DEPE is pipelined and executes one control word per clock cycle.

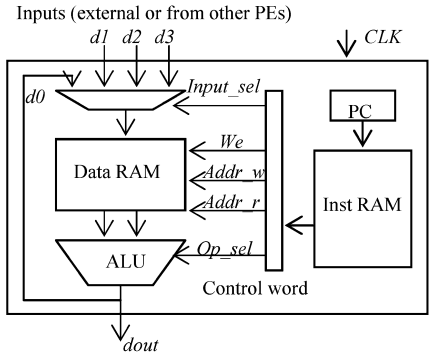



Fig. 3. DEPE architecture.

TABLE III
ASIP AND DEPE RESOURCE USAGE AND CLOCK FREQUENCY (MHz)

| Data RAM | ASIP | | | | DEPE | | | |
|---|---|---|---|---|---|---|---|---|
| | LUTs | DSP | BRAM | Freq | LUTs | DSP | BRAM | Freq |
| 64 | 517 | 1 | 1 | 182 | 341 | 1 | 1 | 198 |
| 128 | 607 | 1 | 1 | 155 | 468 | 1 | 1 | 166 |
| 512 | 424 | 1 | 3 | 162 | 233 | 1 | 3 | 175 |

Table II shows performance results. DEPE is about 8% faster than ASIP due to the faster clock frequency.

### C. Memory Model of DEPE

Since DEPE implements a small part of a physical model, each DEPE contains a local data RAM to store the data of that part. For large models, data would be distributed among multiple DEPEs. There is no centralized global memory in a multi-DEPE approach, so the DEPE approach may scale well.

### D. ASIP and DEPE Resource Usage

Table III shows ASIP and DEPE resource usage. We consider three Data RAM size/implementations: the 64 and 128 words drams are implemented with LUTS, and the 512 words dram is implemented with a block RAM. DEPE uses 50% fewer LUTs, due mostly to eliminating instruction decoding logic.

Compared to a MicroBlaze, the ASIP uses 3$\times$ less LUTs, while DEPE uses 4$\times$ less LUTs. The ASIP and DEPE use 1/3 of the DSP units and use less BRAM units (1 and 3 versus 8). The tradeoff is of course the reduced generality of the ASIP and DEPE, but they are sufficient for ODE solving purposes.

### E. DEPE Compiler

We developed a custom compiler to translate physical model equations to DEPE control words. The input format is shown in Fig. 4(a) (representing a RC lung model in Fig. 1, note R = 1). The input begins with parameter settings (e.g., zero, compliance, and time step constants), initial variable values, a list of external inputs, and then one or more equations. The equation syntax is similar to that used in JSIM [8]. Finally, the input ends with indication of the iterative solution method (Euler, RK2, or RK4), and the solution's time step in seconds.

The compiler transforms the equations based on the method chosen. Fig. 4(b) shows the transformed equations for Euler, as well as for an RK2 solver for illustration. Note the RK2 method calculates the derivative twice per step.

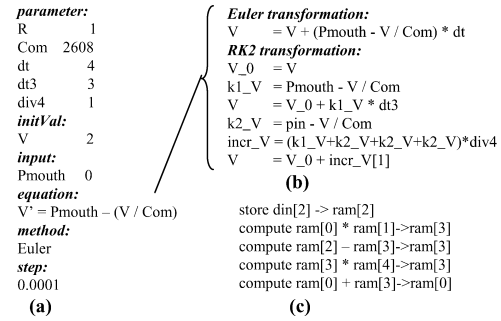



Fig. 4. DEPE compiler: (a) input, (b) transformed equations (Euler and RK2), and (c) generated control words (Euler).

Finally, the compiler generates control words to execute each equation, as in Fig. 4(c) for the Euler transformation. Those words' binaries are stored in Instruction RAM.

The compiler determines the necessary sizes of Data and Instruction RAM. The maximum Data RAM size is set to 512 for DEPE, otherwise the control word's width would surpass 32 bits. Based on our experiments, the 512-word Data RAM can support about 100 ODEs, satisfying our requirement that DEPE implement a small part of a physical model.

### *F. Comparison With Non-FPGA Platforms*

For further comparison purposes, we also implemented the models on several common computing platforms:

1) *Desktop*: C code on a Pentium4 3.0 GHz processor with 8KB L1 and 1MB L2 cache, compiled using gcc with $-$O2 flag.
2) *ARM*: C code on a 450 MHz TI ARM9e embedded processor with 32KB L1 and 128KB L2 cache, compiled using TMS470 compiler with $-$O2 flag.
3) *DSP*: C code on a 700 MHz TI C6424 digital signal processor with 112KB L1 and 128KB L2 cache, compiled using TI C6000 compiler with $-$O2 flag.

We used the same fixed-point computation as on the FPGA for a fair comparison. For fastest ODE solving on processors, we first compared explicit ODE equation calculation with standard libraries using matrix-based approaches [3], [6], but found the explicit approach fastest so used such an approach. ARM and DSP performance are simulated with TI CCS [13] cycle accurate profiler. Performance results appear in Table IV. The ARM, DSP and DEPE showed comparable performance, executing all models $> 10\times$ faster than real-time. Of course, ARM and DSP would be much larger than DEPE on an FPGA and their frequencies would be less. DEPE's similar performance with lower clock frequency is due to its equation solving efficiency via its single-cycle compute operation and large (up to 512-word) Data RAM. The Desktop implementation is clearly fastest, executing all models $> 100\times$ faster than real-time, but of course comes with high cost and power. For reference, we ran the 5 models in Matlab on the desktop, resulting in performance $160\times$ slower than C on the desktop.

Although DEPE's performance is about 10X slower than the desktop, its small resource usage permits hundreds of 100 DEPEs to be placed on the target Virtex6 FPGA. Our recent studies show that a larger Xilinx Virtex6 240T FPGA can hold up to 400 PEs, and runs more than $30\times$ faster than a Pentium4 3.0 GHz desktop for a Weibel 11 generation lung and a Lutchen model with 4000 ODEs. Future work will involve synthesis/compilation for DEPE networks.

TABLE IV
PERFORMANCE (MS) ON NON-FPGA PLATFORMS

| Model | Desktop | ARM | DSP | DEPE |
|---|---|---|---|---|
| W2 (6 ODEs) | 2 | 24 | 15 | 11 |
| L10 (10 ODEs) | 2 | 16 | 15 | 19 |
| W4 (30 ODEs) | 7 | 90 | 81 | 66 |
| Heart (31 ODEs) | 8 | 96 | 75 | 87 |
| L50 (50 ODEs) | 9 | 78 | 74 | 108 |

## V. CONCLUSION

DEPE is a soft-core FPGA processor created specifically for fast and resource-efficient differential equation solving. Such solving on FPGAs is important today because physical models are increasingly implemented on FPGAs to speed up simulations or to support real-time cyber-physical system testing. DEPE's NISC-style architecture uses only two thirds the LUTs of an ASIP and has an 8% faster clock frequency, and is 3–6$\times$ smaller and 4–17$\times$ faster than a MicroBlaze soft-core processor, with performance comparable to off-the-shelf ARM and DSP processors having much faster clock frequencies. DEPE was created as the foundation for future parallel networks of processors on FPGAs to support large physical models. DEPE's programmability enables future instrumentation for profile and debug purposes.

## REFERENCES

[1] K. Atkinson, *Elementary. Numerical Analysis*, 2nd ed. New York: Wiley., 1993.
[2] AutoESL [Online]. Available: http://www.xilinx.com/tools/autoesl.htm
[3] Boost uBLAS 2011 [Online]. Available: www.boost.org/doc/libs/release/libs/numeric
[4] J. C. Butcher, Numerical Methods for Ordinary Differential Equations 0471967580, 2003.
[5] Celoxica 2011 [Online]. Available: http://www.celoxica.com/
[6] Eigen 2011 [Online]. Available: http://eigen.tuxfamily.org
[7] J. A. Fisher, P. Faraboschi, and G. Desoli, "Custom-fit processors: Letting applications define architectures," in *Proc. ACM/IEEE Int. Symp. Microarch. (MICRO)*, 1996, pp. 324–335.
[8] JSIM 2011 [Online]. Available: http://nsr.bioeng.washington.edu/jsim/
[9] F. P. Lutchen, J. R. Primiano, and G. M. Saidel, "A nonlinear model combining pulmonary mechanics and gas concentration dynamics," *IEEE Trans.*, vol. BME-29, pp. 629–641, 1982.
[10] MedGadget, "Supercomputer creates most advanced heart model," *Internet J. Emerging Med. Technol.*, Jan. 2008.
[11] J. Pimentel and Y. Tirat-Gefen, "Hardware acceleration for real time simulation of physiological systems," in *EMBS*, 2006.
[12] M. Reshadi, B. Gorjiara, and D. Gajski, "Utilizing horizontal and vertical Pparallelism using a no-instruction-set compiler and custom datapaths," in *Proc. ICCD*, San Jose, CA, 2005.
[13] TI CCS [Online]. Available: http://focus.ti.com/docs/toolsw/folders/print/ccstudio.html
[14] J. Villarreal, A. Parlk, W. Najjar, and R. Halstead, "Designing modular hardware accelerators in C with ROCCC 2.0," in *Proc. FCCM*, Charlotte, VA, 2010, pp. 127–134.
[15] E. R. Weibel, *Morphometry of the Human Lung*. Berlin, Germany: Springer-Verlag, 1963.
[16] R. L. Winslow *et al.*, "Mechanisms of altered excitation-contraction coupling in canine tachycardia-induced heart failure. II. Model studies," *Circ. Res.*, vol. 84, pp. 571–586, 1999.
[17] M. Yoshimi, Y. Osana, T. Fukushima, and H. Amano, "Stochastic simulation for biochemical reactions on FPGA," in *FPL*, 2004.