Architectural Implications of Nanoscale-Integrated Sensing and Computing

THE AUTHORS EXPLORE NANOSCALE SENSOR PROCESSOR (NSP) ARCHITECTURES. THEIR DESIGN INCLUDES A SIMPLE ACCUMULATOR-BASED INSTRUCTION-SET ARCHITECTURE, SENSORS, LIMITED MEMORY, AND INSTRUCTION-FUSED SENSING. USING NSP TECHNOLOGY BASED ON OPTICAL RESONANCE ENERGY TRANSFER LOGIC HELPS THEM DECREASE THE DESIGN'S SIZE; THEIR SMALLEST DESIGN IS ABOUT THE SIZE OF THE LARGEST-KNOWN VIRUS.

Constantin Pistol
Wutichai Chongchitmate
Christopher Dwyer
Alvin R. Lebeck
Duke University

Understanding molecular-scale phenomena is a critical component of many scientific disciplines. The ability to retrieve nanoscale information from within macroscale systems is particularly useful in biological fields, where the diversity of molecular components and interaction dynamics within a cell make it difficult to monitor and quantify the underlying processes. Current methods rely on custom-designed molecules called *molecular probes* that alter their observable properties to acquire real-time information about nanoscale phenomena.

Molecular probes are important members in the biological scientist's toolbox; however, they generally function as standalone sensors. Furthermore, their use requires costly equipment and highly specialized training, and often the experiments span several days. If researchers could find a low-cost, efficient way to monitor complex, molecular-scale biological processes, it could greatly expand current abilities to track and diagnose diseases. For example, people could have at-home early disease detection with a low-cost device capable of monitoring important markers of bioactivity and cellular health, such as concentrations of specific proteins or small messenger ribonucleic acid (mRNA) molecules.[1]

We often use computing to monitor complex processes or automate tasks that require expert training. However, biological-scale computing represents a new domain for computing with greatly different constraints from traditional computing systems. For example, this new computing domain requires the ability to diffuse through a volume of small molecules, computing in and sampling the same local environment as a molecular probe (for example, in the homogenized contents of a cell). This size requirement excludes current complementary metal-oxide semiconductor (CMOS) solutions because large (several microns) silicon chips don't diffuse freely. Although a CMOS processing core connected to biosensors[2] could

Published by the IEEE Computer Society

0272-1732/10/$26.00 © 2010 IEEE

read and process chemical information, it wouldn't be able to automate molecular probe applications because of its large aggregate size.

Fortunately, recent advances in nanotechnology could provide the appropriate capabilities for molecular-scale biologically compatible computing. This article presents the concept of a *nanoscale sensor processor* (nSP) that addresses the aforementioned challenges by integrating molecular probe sensors and molecular-scale digital logic.[3]

An nSP is a nanoscale-sized system that can sense, process, store, and communicate molecular information. A generic nSP has several components: a sensor array for environment monitoring, a simple processor core, a small memory for state and programs, and a communication device for information transfer to the macroscale.

Each element of the sensing array is a molecular probe designed to detect the presence of a specific target molecule—called an *analyte*—through chemical bonding. The size restriction for molecular probe applications requires a computational substrate that can cost-effectively support meaningfully complex circuits with nanometer feature sizes and provide a sensing ability. For a sample of related work, please see the sidebar.

Related work

The most closely related work to ours falls within the general realm of amorphous computing,[1] which is predicated on the existence of large numbers of inexpensive devices with limited computational ability, limited memory capacity, and limited communication ranges. The set of potential applications for amorphous computing is vast, ranging from smart paint to in-vivo computation for biological applications. Our work also has similarities to the broad area of sensor networks[2] and ultra-low-power sensor processors.[3] Other work explores novel molecular logic, but currently only presents individual gate functionality.[4]

References

1. H. Abelson et al., ''Amorphous Computing,'' *Comm. ACM,* vol. 43, no. 5, 2000, pp. 74-82.
2. J. Hill et al., ''System Architecture Directions for Networked Sensors,'' *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM Press, 2000, pp. 93-104.
3. B. Zhai et al., ''A 2.60pj/inst Subthreshold Sensor Processor for Optimal Energy Efficiency,'' *Proc. Symp. Very Large-Scale Integration Circuits,* IEEE Press, 2006, pp. 154-155.
4. A.P. de Silva and N.D. McClenaghan ''Molecular-Scale Logic Gates,'' *Chemistry—A European Journal,* vol. 10, no. 3, 2004, pp. 574-586.

Enabling technology

We propose a new nanoscale technology for nSPs based on single-molecule optical devices called *chromophores*. In isolation, a given chromophore absorbs photons of a specific wavelength and emits photons at a different, lower-energy wavelength. However, when we place appropriate chromophores a few nanometers apart, one chromophore can transfer an absorbed photon's energy to a neighboring chromophore through a process called *resonance energy transfer* (RET). Scientists frequently use this process for molecular-scale sensing (for example, molecular beacons or molecular rulers).[4] But RET also provides a theoretical foundation, which we're exploring, for creating pass gates (both inverting and noninverting) using four chromophores per gate.[5] These gates form a complete Boolean logic set we call *RET logic*.

To fabricate RET logic, we use deoxyribonucleic acid (DNA)-based self-assembly to place chromophores within specified distances. The specific DNA nanostructures we use are grids in which we can place two pass gates and one wire crossover per grid vertex. We can assemble the grids hierarchically to create large arrays of pass gates—the nanoscale equivalent to a sea of gates.

Resonance energy transfer (RET)-based circuits

We designed several RET-logic circuits and performed manual layout on DNA grids. The layouts in Figure 1 show potential implementations of common RET-logic circuit components (memory, decoder, and 1-bit full adder cells) on a 20-nanometer (nm) pitch DNA substrate. The circuits utilize inverting and noninverting pass gates; we annotated wires with the frequency of their specific optical signal ($a > b > c > d > g$).

As a proof of concept we assembled and characterized a wired-OR gate on grids in our laboratory. We assembled the gate using three chromophores—two for the

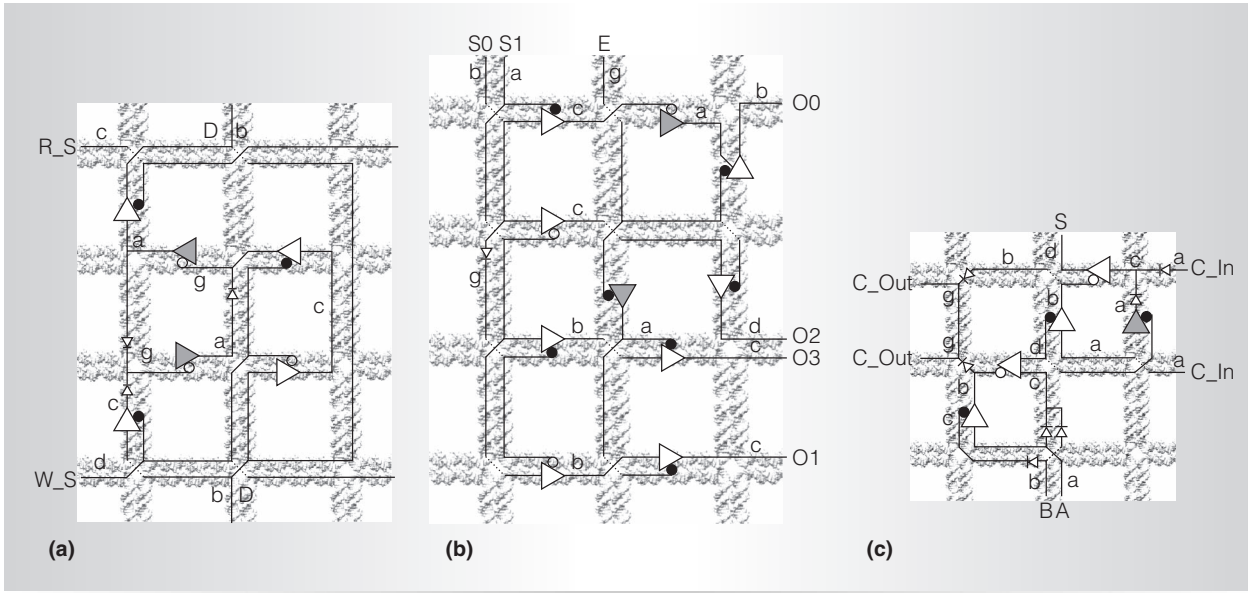
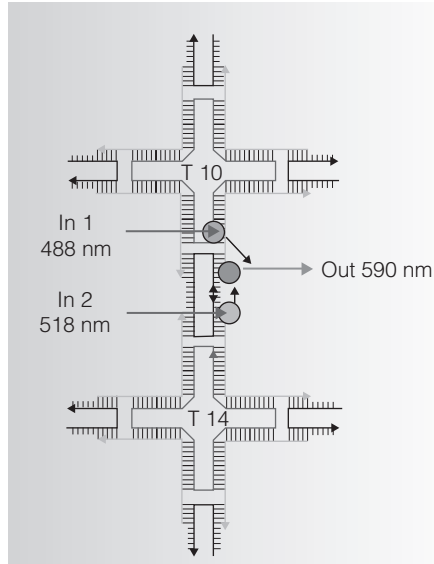


Figure 1. Resonance energy transfer (RET)-logic circuit layouts on a deoxyribonucleic acid (DNA) substrate with 20-nanometer (nm) grid spacing. A static RAM (SRAM) cell (a) that's 60 × 40 nm, a decoder cell (b) that's 60 × 40 nm, and a 1-bit full adder cell (c) that's 40 × 40 nm.



Input 1 (488 nm)	Input 2 (518 nm)	Resonance energy transfer (RET) output (590 nm)
Off	Off	0
On	Off	39
Off	On	31
On	On	70

Figure 2. Observed RET output (in optical fluorescence counts) from the wired-OR gate with 488-nm (in 1) and 518-nm (in 2) excitation.

input signals (Oregon Green and Alexa Fluor 535) and one for the output signal (Rhodamine Red) attached to the grid (as Figure 2 shows). The two inputs are excited by wavelengths of light at 488 nm and 518 nm for Oregon Green and Alexa Fluor 535, respectively. The input chromophores undergo RET with the Rhodamine Red output chromophore, which has an emission peak at 590 nm.

RET-based sensing

RET sensors are devices that transduce a molecular recognition event—such as the binding of a target analyte—into a modulation of their optical RET properties. A wide range of biological RET sensors are currently available.[4] The fundamental property of these sensors is the dependence of RET on chromophore separation. RET only occurs when chromophores are sufficiently close (a few nanometers), and decreases as the distance increases. For sufficiently separated chromophores, RET doesn't occur. We can design sensors to enhance RET or prevent RET when a molecular recognition event occurs. Also, because RET sensors use the same interchromophore energy transfer as RET logic, they can interface directly with RET-logic circuits.

Understanding application requirements

One of an nSP's critical design requirements is that it be small enough to diffuse through small volumes. A typical red blood cell width is approximately 6 to 8 micrometers (μm), and we envision applications that require operation within a cell. The RET-logic nanotechnology only provides the potential to create appropriately sized nSPs. To meet severe size constraints, even RET-logic systems must be carefully architected. To achieve this, we need to understand the computational requirements of the biological applications (see the "Nanoscale sensor processor applications" sidebar).

Through the inspection of target applications, we can extract several important characteristics that influence architectural designs, including

- long time scales,
- accumulating values,
- waiting for an event, and
- processing groups of sensor values as an aggregate.

Each application characteristic influences the nSP architecture, either individually or when combined with other characteristics. We discuss each characteristic and its qualitative influence on the architecture. First, the long time scales (seconds to minutes) of biological applications imply that we don't need a high-performance processor core (so no superscalar, out-of-order, deep pipeline). Instead, the architect can trade area for time using a simple processor core with complex operations synthesized in software (for example, multiplying and dividing).

The second characteristic is that many of the applications accumulate values over time either by counting events, averaging, or integrating values that are monotonically increasing. This has several implications for the architecture. First, we can generally perform the computations using fixed-point arithmetic, avoiding the need for floating-point hardware. Second, the accumulated value might require a larger range than the input value, leading to datapath components wider than the memory or sensor. Third, accumulation is common enough that we can justify architectural support to help reduce code size.

The next application characteristic—waiting for an event—is similar to accumulating a value, because the application implements it by reading a sensor value within a loop and either incrementing a counter (accumulating) or checking to see if the sensor value changed (waiting for an event). Like accumulation, waiting for an event is sufficiently common across applications that the architecture should provide support to reduce code size.

Finally, processing individual sensors as a group implies that the architecture might not need to support access to individual sensor values. Instead, it could provide support for processing multiple sensor values as a single entity.

A nanoscale sensor processor

Our nSP architecture is a simple, accumulator-based processor with a small amount of addressable memory and number of sensors. Using a single accumulator reduces the processor core complexity and allows short, 4-bit opcodes to support the common recurring operations in nSP applications. Our standard nSP design can address up to 256 4-bit words in a unified instruction, data, and sensor memory space. Instructions vary in length (4 bits or 12 bits) to decrease the application code's memory footprint. A variant of our architecture, called

Nanoscale sensor processor applications

To illustrate how diversely we can apply this technology, we selected several target applications that we believe are representative of a nanoscale sensor processor's (nSP's) potential. Examples include

- early disease detection,
- custom multianalyte molecular probes,
- monitoring complex biological scale processes,
- molecular kinetics analysis, and
- imaging below the diffraction limit.

Our goal is to understand the applications' requirements such that we can architect an appropriate nSP system. To achieve this we consider two aspects of each application: which algorithm to use and the timing constraints. The algorithm specifies, at a high level, the computation to perform, and we use it to determine which primitive operations to support. The timing constraint sets bounds, both upper and lower, on the latency of a primitive operation or an entire computation.

For many biological applications, the expected time scales are in the range of seconds to minutes.[1] We generally determine these times by either chemical reaction rates or molecular interaction rates. For example, sensors can be either reversible or irreversible. For irreversible sensors, once the analyte binds to the sensor it's bound forever. In contrast, for reversible sensors the analyte will eventually detach from the sensor. We determine the time scales for irreversible sensors using the analyte's binding rate. We can characterize a reversible sensor's time-dependent behavior by two parameters: a period and a dwell time. The period is the time between two distinct binding events, and the dwell time is the time an analyte stays bound to the sensor after the binding event. The period and dwell time of a sensor often represent average values for a probabilistic distribution of times.

We present pseudocode for each of the applications in Figure A with the goal of demonstrating that an nSP can provide capabilities beyond a single sensor. This pseudocode assumes the ability to read a specified sensor—for example, `read_sensor(name)`—that targets a particular analyte and returns a Boolean value ("true" when the analyte is bound to the sensor, and "false" otherwise). We also assume the ability to output an integer value.

Reference

1. I. Golding et al., "Real-Time Kinetics of Gene Activity in Individual Bacteria," *Cell,* vol. 123, no. 6, 2005, pp. 1025-1036.

```
while (true) {
  sample = read_sensor(P)
  if (sample != last_sample) {
    count += sample
    last_sample = sample }
  if (send_data = true) {
    output(count)
    count=0}}
```
(1)

```
wait until (read_sensor(A) = true)
wait until (read_sensor(B) = true
and
   read_sensor(C) = true)
wait until (read_sensor(D) = true
and
   read_sensor(A) = false)
output(true)
```
(2)

```
while (true) {
  sample = 0
  for k = 1 to M {
    sample += read_sensor()}
  d[last] = sample
  y = 0
  for k = 1 to N {
    y += c[k]*d[(last+k)%N]}
  last = (last+1)%N
  output(y) }
```
(3)

```
while (true) {
 sample = 0
 for k = 1 to M {
  sample += read_sensor()}
 output (sample) }
```
(4)

```
while (true) {
 if (send_data = true)
  for i = 1 to N {
   output(sensor_read(i))}}
```
(5)

Figure A. Pseudocode of target applications representing the potential of a nanoscale sensor processor (nSP): Pathogen counting (1), multianalyte probe (2), finite impulse filter (3), kinetic analysis (4), and imaging (5).

Tiny, is designed for the smallest nSPs, with a total memory space of only 8 bytes (16×4 bits).

An external light source performs input to the nSP, illuminating the nSP at specified wavelengths. Similarly, the nSP emits at predefined output wavelengths observed by an external photodector.

Integrated sensing

The sensor-centric nature of nSP applications means that the interface between processing and sensing plays an important role in system design. Because the nSP uses RET for sensing and computing, we can directly integrate the sensing mechanisms into the system design. Specifically, we can augment certain SRAM cells with appropriate sensing mechanisms that force the memory location to the value "1" (or "0" as needed). A technological requirement is that the sensor's active area must fit within the confines of an SRAM cell. We could design these environmentally modified memory locations to provide a set of memory-mapped sensors, or we can interleave sensor-augmented SRAM cells with standard SRAM cells, including those used for instruction opcode bits. For example, we could modify an unconditional jump (JMP) opcode into a no-operation-performed (NOP) opcode using a sensor-binding event. Similarly, we could modify instruction operands using the sensing mechanism to change an arithmetic operand or a branch target. We call this technique *instruction-fused sensing* (IFS).

IFS provides a unique opportunity for hardware and software codesign to improve code density. With only 8 to 128 bytes of memory available, judicious use of instructions is paramount to providing sufficient computational abilities. The simple task of querying a sensor to determine if a specific protein is present can require several bytes of instruction memory to load a value and compare it for branching. Instead, we could use a single-branch instruction that changes to a NOP (and escape) when the protein binds.

Instruction-set architecture

Our nSP instruction-set architecture (ISA) supports several common memory, arithmetic, and control transfer instructions. We include each piece of instruction in the ISA because it directly supports common operations in one or more applications. Note that the ISA lacks several instructions that programmers often consider standard, such as subtract, multiply, and divide. We can synthesize these operations and provide instructions if needed. For brevity, however, for now we only discuss our standard nSP ISA.

We designed several nSP instructions to reduce application code size. The increment-implicit (INCI) instruction—which increments an 8-bit value using implicit addressing (PC relative)—is useful for accumulating single-bit sensor values and for control loops (which are recurring operations in the nSP applications). Several instructions also exploit IFS to reduce code size. In particular, JMP, branch on not zero (BNZ), INC, and INCI all can use IFS. When declared sensitive to an analyte A, these instructions turn into NOPs when that analyte binds to the appropriate memory location. For example, INC (!A) means the accumulator increments only if the analyte A isn't present, or else it's a NOP and has no effect.

To facilitate this mechanism, we encode a single-bit difference between these instructions' opcode and the NOP opcode. When a sensor fuses to this bit location, the opcode value depends on the sensor's value, as determined by an analyte's presence or absence. This requires careful hardware and software codesign to ensure the proper alignment of code and sensors in the memory space.

Instruction operands—either immediate values or addresses—are 8 bits long, the same as the data granularity of loads and stores. This allows for easy address manipulation, but most importantly, it increases the numerical range for operations performed on sensor data. A common pattern in the nSP target applications is that individual sensor data accumulates over time before being further processed, with the option of longer accumulation times being more desirable. Native support for 8-bit arithmetic increases the accumulation time interval from 16 to 256 cycles, while avoiding the severe code size penalty that results from emulation with 4-bit instructions. The accumulator is

```
0   JMP (!send) 6  // if not sending, count
3   OUTCLR 13      // send data and clear
                   // counter
6   JMP (!A) 18    // if no analyte is bound
                   // go to 18
9   BNZ 0          // analyte is bound,
           // if acc is 0 go to next cycle
12  INCI   // increment count, update acc
15  JMP 0  // jump to next cycle
18  CLR    // clear acc (make last
           // event "not bound")
19  JMP 0  // jump to next cycle
```

Figure 3. An 8-bit event counter (pathogen counting).

16 bits for extended dynamic range in applications such as finite impulse result (FIR) filters, in which the accumulator averages or multiplies double word samples through shift-and-add. The system can access the high 8 bits of the accumulator through explicit shifts into the low 8 bits.

Defining only 8-bit-wide memory operations could prove restrictive given sensors' single-bit granularity. Indeed, accessing an individual sensor value through a wide load can require additional bit masking and shifting. We argue, however, that in cases requiring single-bit sensor values, it's generally more efficient to use the compact IFS mechanism. An analysis of the target application algorithms shows three distinct cases in which sensors are individually queried for their value:

1. when the branch outcome depends on an individual sensor value,
2. when the arithmetic result depends on an individual sensor value, and
3. for the output of sensor values.

We can use IFS to address the first two cases. For the third, we can bundle values for output and separate values at a remote receiver.

Evaluation

As we evaluate the nSP's architecture, it's important to note that although we designed it primarily to meet the requirements of the new biological-scale computing domain, the architecture is generic. Users can implement it in several different nanotechnologies that provide both sensing and digital logic. Size is a crucial constraint for an nSP architecture, as any device must be capable of diffusing through small volumes. Our architecture also provides a set of novel mechanisms (for example, IFS) that allow compact implementations of important biological applications.

Node size

We estimate the area of nSP implementations using the RET-logic layouts and characteristics previously described. Our preliminary layout indicates that the standard nSP implementation with 128 bytes of memory requires 2.5 × 2.5 μm and the Tiny nSP implementation with 8 bytes of memory requires 800 × 800 nm. With a surface area comparable to the largest known virus,[6] the Tiny nSP could diffuse in the same environments as the virus (such as a cell).

Program size

We implemented the target applications for both the standard and Tiny nSP architectures. The memory requirement for the standard implementation (code and data) ranges from 9 to11.5 bytes with the exception of the moving average filter, which uses 59 bytes. The Tiny implementations all fit within the limited 8-byte memory space. For most of the applications, the memory footprint depends on the number of monitored sensors; more sensors require more memory space.

The nSP program in Figure 3 implements the pathogen-counting algorithm, which monitors a single, reversible, pathogen sensor and outputs the number of 1 to 0 transitions observed since the previous report. The program uses an 8-bit counter to accumulate up to 255 events between output reports. The total memory footprint is 11 bytes and a saturating counter version (omitted for brevity) uses 14.5 bytes. We achieved this compact code by using IFS for event detection whenever a pathogen was present and we needed to transmit results. The INCI instruction also provides compact accumulation for the counter.

The remaining applications all require less than the 128 bytes available on the standard nSP. Our FIR implementation—which

uses a moving average—is the largest, most complex application and requires just under 60 bytes. Kinetic analysis and imaging each require around 10 bytes.

Figure 4 shows the Tiny nSP implementations for four applications. The FIR is too complex to fit within the 8-byte limitation. The main cost of using Tiny nSPs is that it reduces the dynamic range for counting and accumulation from 8 to 4 bits and it also greatly limits the number of sensors that a program can monitor. The fundamental advantage of Tiny nSPs is their diminutive size: they can diffuse through and sample molecular environments that are inaccessible to larger standard nSPs.

Pathogen counting	Kinetic analysis
0 JMP (!send) 4	0 INCI (!A)
2 OUTCLR 9	2 INCI (!B)
4 JMP (!A) 12	4 INCI (!C)
6 BNZ 0	6 INCI count
8 INCI	8 BNZ 0
10 JMP 0	10 OUTCLR 1
12 CLR	12 OUTCLR 3
13 JMP 0	14 OUTCLR 5
Imaging	**Multianalyte probe**
0 INCI count	0 JMP (!A) 0
2 OUT 1	2 JMP (!B) 2
3 JMP 0	4 JMP (!C) 2
5-15 SENSORS	6 JMP (!D) 6
	8 JMP (E) 6
	10 OUT 2

Figure 4. Tiny nanoscale sensor processor (nSP) programs using 8 bytes of memory space.

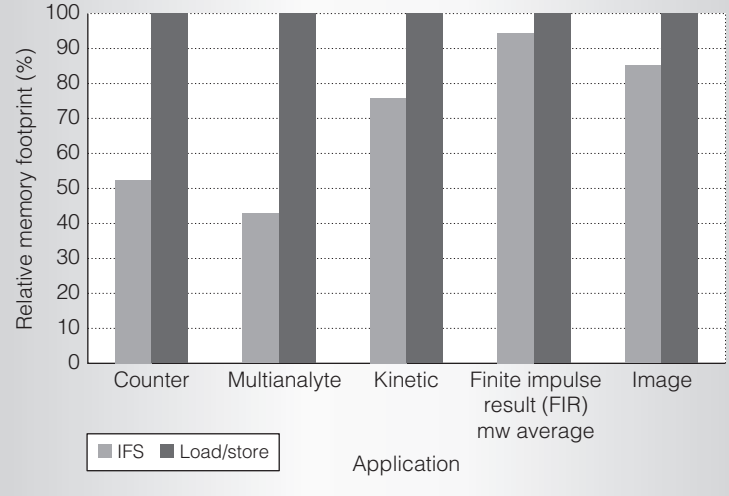


Figure 5. Impact of using instruction-fused sensing (IFS) on the memory requirement of target nSP applications (compared to using a load-and-store approach).

Impact of instruction-fused sensing

A single IFS instruction can replace several non-IFS operations and significantly reduce applications' memory requirement. Figure 5 shows the total memory required by IFS relative to non-IFS (load and store) implementations of our applications for the standard nSP. The IFS reduces the footprint to 58 percent (in the case of the multisensor analyte probe) and 5 percent (for the single-sensor, processing-intensive FIR). However, when using IFS, instructions become statically bound to sensors, preventing conventional code size-reducing techniques, such as procedures or loops. This trade-off increases in importance when monitoring many individual sensors in a single program.

Application simulation results

We verified the expected output of nSP applications using a simple cycle-level nSP functional simulator augmented with chemical environment and sensor–analyte interaction simulation. The simulator performs memory access (for a 4-bit word, including sensor read) and arithmetic operations in a single cycle. The total number of cycles per instruction varies between two and six depending on the instruction length and number of memory accesses.

The environment simulation models time-varying concentrations of analytes and the corresponding binding and dissociation events for each nSP sensor. We modulate the binding event probability with binding and dissociative rate constants that we explicitly specify for all distinct sensor–analyte pairs.

For each application, we initialized the nSP with the appropriate program, simulated a chemical environment characteristic for that application, and followed the program

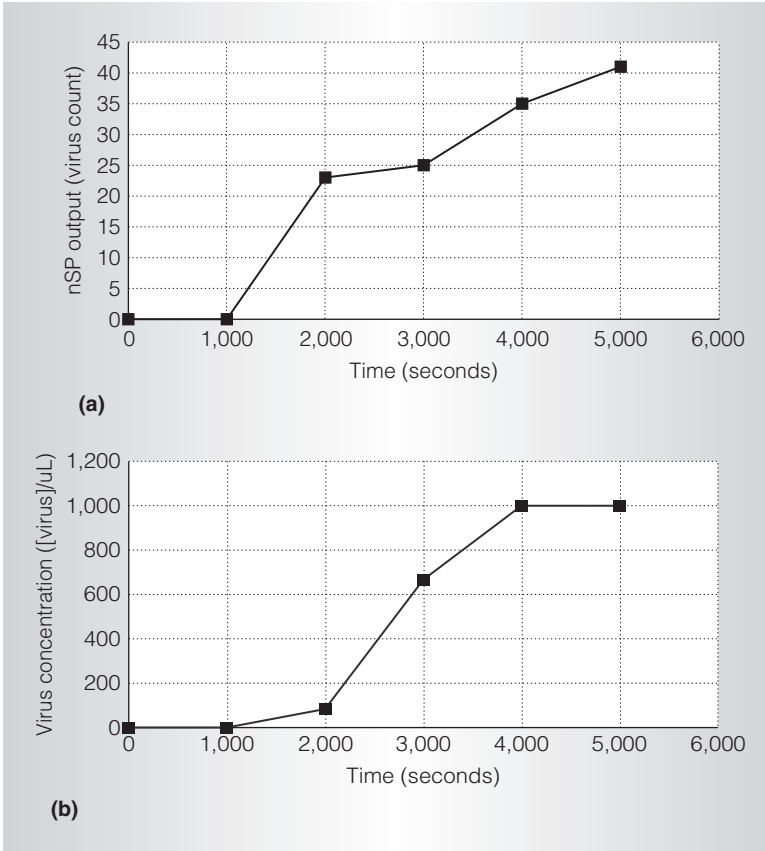


Figure 6. Influenza-A virus counting simulation. The nSP node output (a) follows the virus presence (b).

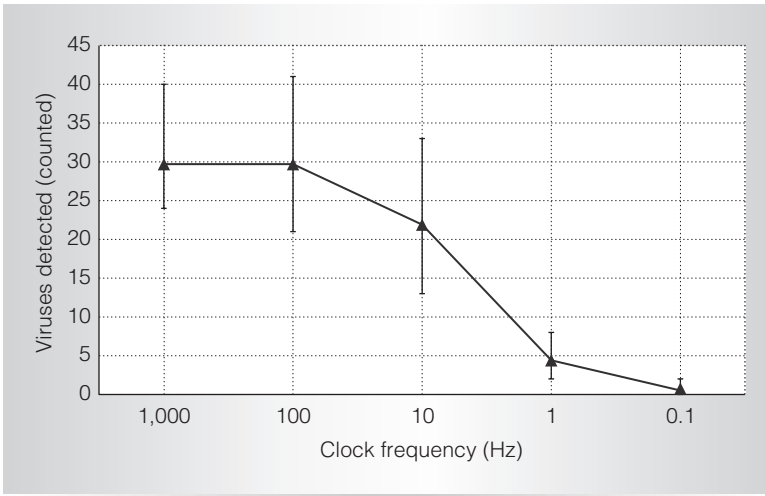


Figure 7. The pathogen-binding kinetics determine the minimum nSP clock rate before counting becomes inaccurate because of missed events.

output in time. For brevity, we discuss only the pathogen counting and multianalyte probe applications.

Pathogen counting. We simulated an influenza virus environment using the published pathogen period and dwell time for antihemagglutinin-based sensors.[7] Figure 6a shows the output of a standard nSP running the pathogen-counting code from Figure 3 in the presence of a time-varying pathogen concentration (see Figure 6b). The nSP clock cycle time is 100 Hz, and the program requests output via an external optical send signal every 1,000 seconds.

When requested, the program outputs the number of observed pathogen-binding events since the last transmission. As expected, the output's value correlates with pathogen concentration and shows variations because of stochastic (single-molecule) sensor binding. Someone observing this trend could use this data to diagnose an infection.

The time scale of the analyte–sensor interaction determines the minimum nSP clock cycle time. If the program samples sensors too slowly, it could miss binding events. In this experiment, the critical sampling threshold is less than one second (the average dwell time for this pathogen–sensor combination). Therefore, the nSP clock rate must be fast enough to ensure that the program samples sensors frequently enough. Figure 7 shows the output of the virus-counting application running at various nSP clock rates (averaged over 10 queries taken 1,000 seconds apart). We held constant the virus concentration and the window of time over which the nSP counts binding events. Given our multicycle nSP, the clock frequency must be 100 Hz or greater to avoid a misdiagnosis. At lower clock rates, the program misses pathogen-binding events, as reflected by the decreasing counter value.

Multianalyte probe. We model five generic analytes (labeled A through E) and execute the molecular probe application code that detects the A, then B and C, then D and not E sequence of events. Figure 8 shows the input concentration of each analyte and the program output (out) over time.

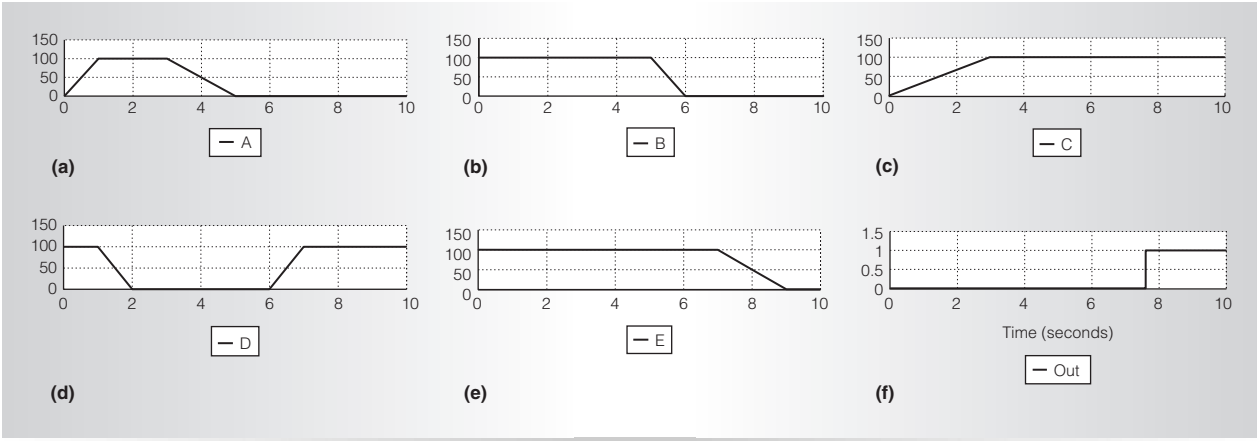


Figure 8. Multianalyte probe simulation. The node output (out) depends on the sequence of analytes (A, B, C, D, and E).

The result emphasizes the probe's local, single-molecule sensing characteristics. Even though there's still some concentration of analyte E present in the system, the program asserts its detection output (just before the 8's mark). The reason is the output decision's stochastic nature, based on instantaneous nSP-local values of sensors, which will probabilistically encounter time intervals with no bound analyte, even if globally the analyte is still present. This could be eliminated by observing analytes over a longer window of time to determine their presence or absence.

The work presented in this article represents our first steps toward developing biological-scale computing systems. In the future, we hope to advance our research by demonstrating integrated logic and sensing, exploring additional RET-logic devices and wires, and designing new DNA nanostructures that can provide more compact nSP designs. MICRO

Acknowledgments

This work is supported in part by the US National Science Foundation (grants CCF-0829911 and CCF-0702434), Agilent, and IBM.

References

1. D. Endy, ''Foundations for Engineering Biology,'' *Nature,* vol. 438, no. 7067, 2005, pp. 449-453.
2. T. Vo-Dinh et al., ''DNA Biochip Using a Phototransistor Integrated Circuit,'' *Analytical Chemistry,* vol. 71, no. 2, 1999, pp. 358-363.
3. C. Pistol et al., ''Architectural Implications of Nanoscale Integrated Sensing and Computing,'' *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* ACM Press, 2009, pp. 13-24.
4. J.R. Lakowicz, *Principles of Fluorescence Spectroscopy,* Kluwer Academic/Plenum, 1999.
5. C. Pistol et al., ''Nanoscale Optical Computing Using Resonance Energy Transfer Logic,'' *IEEE Micro,* vol. 28, no. 6, 2008, pp. 7-18.
6. M. Suzan-Monti et al., ''Genomic and Evolutionary Aspects of Mimivirus,'' *Virus Research,* vol. 1, no. 117, 2006, pp. 145-155.
7. F. Patolsky et al., ''Electrical Detection of Single Viruses,'' *Proc. Nat'l Academy of Sciences,* Nat'l Academy of Sciences, vol. 101, no. 39, 2004, pp. 14017-14022.

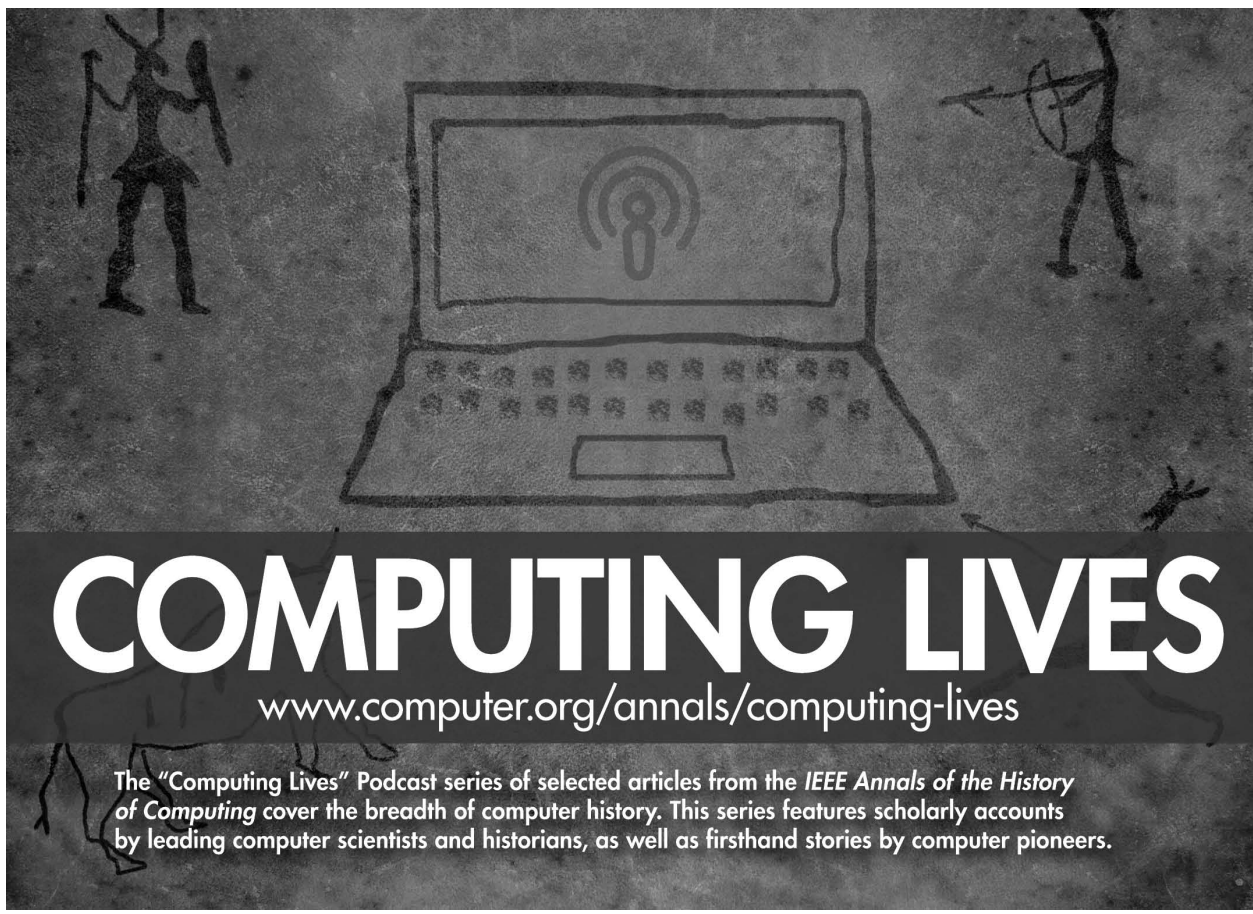
Constantin Pistol is working with industry. His research interests include computer architecture and molecular-scale computation. Pistol has a PhD in computer science from Duke University.

Wutichai Chongchitmate is working on a BS degree in computer science and mathematics at Duke University.

Christopher Dwyer is an assistant professor of computer science and of electrical and computer engineering at Duke University.

His research interests include self-assembly, DNA, and the development of circuit technologies for molecular-scale fabrication processes. Dwyer has a PhD in computer science from the University of North Carolina at Chapel Hill. He is a senior member of the IEEE and a member of the ACM.

Alvin R. Lebeck is a professor of computer science and of electrical and computer engineering at Duke University. His interests include architectures for emerging nanotechnologies, multicore processors, memory systems, and energy-efficient computing. Lebeck has a PhD in computer science from the University of Wisconsin–Madison. He is a senior member of the IEEE and a member of the ACM.

Direct questions and comments to Alvin R. Lebeck, Duke University, D308 LSRC, Department of Computer Science, Box 90129, Durham, NC 27708; alvy@cs.duke.edu.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.