Architectural Implications of Nanoscale Integrated Sensing and Computing

Constantin Pistol   Wutichai Chongchitmate   Christopher Dwyer[Ŧ]   Alvin R. Lebeck

Department of Computer Science
[Ŧ]Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708
costi@cs.duke.edu, wutichai.chongchitmate@duke.edu, dwyer@ece.duke.edu, alvy@cs.duke.edu

Abstract

This paper explores the architectural implications of integrating computation and molecular probes to form nanoscale sensor processors (nSP). We show how nSPs may enable new computing domains and automate tasks that currently require expert scientific training and costly equipment. This new application domain severely constrains nSP size, which significantly impacts the architectural design space. In this context, we explore nSP architectures and present an nSP design that includes a simple accumulator-based ISA, sensors, limited memory and communication transceivers. To reduce the application memory footprint, we introduce the concept of instruction-fused sensing. We use simulation and analytical models to evaluate nSP designs executing a representative set of target applications. Furthermore, we propose a candidate nSP technology based on optical Resonance Energy Transfer (RET) logic that enables the small size required by the application domain; our smallest design is about the size of the largest known virus. We also show laboratory results that demonstrate initial steps towards a prototype.

Categories and Subject Descriptors C.1.0 [**Computer System Organization**]: Processor Architectures – General
General Terms Design, Performance

1. Introduction

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 tool box, however they generally function as standalone sensors. Furthermore, their use requires costly equipment, highly specialized training and often the experiments span several days [23]. These limitations prevent the application of molecular probes in monitoring complex biological processes with low cost. For example, at home early disease detection could be achieved with a low-cost device capable of monitoring important markers of bioactivity and cellular health, such as concentrations of specific proteins or small messenger RNA (mRNA) molecules [6]. The challenge is to develop techniques that provide low cost, efficient monitoring of complex molecular scale biological processes.

Computing is often used to monitor complex processes or automate tasks that require expert training. However, biological scale computing represents a new domain for computing with very 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 (e.g., the homogenized contents of a cell). This size requirement excludes current CMOS solutions since large (several microns) silicon chips do not diffuse freely. Although a CMOS processing core connected to bio-sensors [24] could read and process chemical information, it would not be able to automate molecular probe applications because of its large aggregate size. Fortunately, recent advances in nanotechnology may provide the appropriate capabilities for molecular scale biologically compatible computing.

This paper presents the concept of a nanoscale sensor processor (nSP), which addresses the above challenges through the integration of molecular probe sensors and molecular scale digital logic. An nSP is a nanoscale sized system that can sense, process, store and communicate molecular information. A generic nSP has several components: sensor array for environment monitoring, simple processor core, 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 sensing ability. There are many examples of nanotechnology (e.g., carbon nanotubes[2, 27], silicon nanorods[12], DNA/enzymatic reactions[3]) that provide both a logic system and sensing capability. In this paper we focus on one specific technology—called Resonance Energy Transfer logic—as the basis for our nSP design, see Section 2. However, we note that much of our analysis and architectural design is independent of the specific technology.

The architectural design space of an nSP is strongly influenced by the limited size and by the target application characteristics. Section 3 presents several applications that vary in complexity and discusses the characteristics that influence nSP architecture design. These characteristics include: 1) long time scales, 2) accumulating values, 3) waiting for an event, and 4) processing groups of individual sensor values as an aggregate.

Section 4 presents our nSP architecture, a simple accumulator data path with variable length instructions (either 4 bits or 12 bits). We present two different designs that implement the same base ISA, but differ in operand width and the amount of memory provided. The Standard design provides 256 4-bit memory locations while the Tiny implementation provides only 16 4-bit memory locations. We also introduce the concept of instruction-fused sensing that exploits unified compute/sensing technologies to enable direct environmental modification of instruction bits, and thus reduce overall code size.

We evaluate our nSP designs in Section 5 using simple models and a custom simulator. Our results show that the Standard design occupies approximately 2.5μm x 2.5μm while the Tiny implementation requires approximately 800nm x 800nm, comparable to the largest known virus. We demonstrate that our target applications can fit within the restricted memory space of the Standard nSP, and that four of our five applications can be implemented in the 8 Bytes of memory on a Tiny nSP. We also explore the design tradeoffs in our nSP instruction set and use simulation to demonstrate the execution of applications in a time varying chemical environment.

Related work is discussed in Section 6 and we conclude in Section 7.

2. Enabling Technology

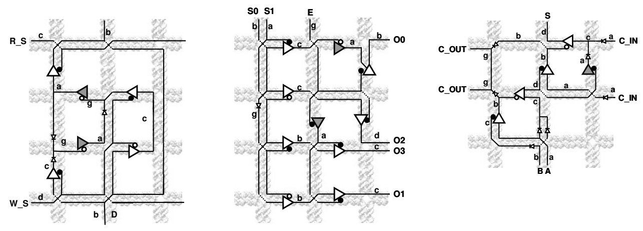
The integration of computation and sensing in a nanoscale package imposes challenging requirements on the underlying technology and manufacturing of nSPs. To date there is no clear winner in the field of nanotechnology for computing or sensing. Carbon nanotube and ring-gated nanorod FETs are hopeful candidates, but obtaining control over the precise device length and precise placement for arbitrary patterns remain open challenges.

We propose using a new nanoscale technology for nSPs based on single-molecule optical devices called chromophores [5, 19]. In isolation, a given chromophore absorbs photons of a specific wavelength and emits photons at a different, lower energy, wavelength. However, when appropriate chromophores are placed a few nanometers apart the energy of an absorbed photon can be transferred to a neighboring chromophore through a process called Resonance Energy Transfer (RET). This process is frequently used for molecular scale sensing (e.g., molecular beacons or molecular rulers) [14]. RET also provides a theoretical foundation, that we are exploring, for the creation of pass gates (both inverting and non-inverting) using four chromophores per gate. These gates form a complete Boolean logic set we call RET-logic.

A key requirement for RET-logic is to place unique chromophores within a few nanometers of each other. Unfortunately, creating such devices using conventional top-down fabrication techniques is costly and increasingly complex. Creating sophisticated circuits by placing individual atoms requires more energy and time than exploiting chemical self-assembly techniques. Furthermore, self-assembly enables fabrication through composition and hierarchies. Different types of molecules can be fabricated independently using the most cost-effective method for each type of molecule. Larger motifs can then be created through the composition of heterogeneous molecules.

RET-logic uses DNA-based self-assembly as the fabrication method to place chromophores within specified distances. The specific DNA nanostructures we use are grids where we can place two pass gates and one wire crossover per grid vertex. The grids can be hierarchically assembled to create large arrays of pass gates—the nanoscale equivalent to a sea-of-gates.

The next section describes some of the characteristics and tradeoffs of RET-logic computational elements integrated on self-assembled DNA substrates, followed by an overview of current experimental progress towards an nSP prototype.



a) SRAM Mem. Cell 60nmx40nm b) 2 to 4 Decoder 60nmx40nm c) 1-bit Full Adder 40nmx40nm

Figure 1. RET logic circuit layouts on DNA substrate. Grid spacing is 20nm.

2.1 RET-logic Circuits

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

Each RET gate has an operational energy cost: the output signal is at a longer wavelength than the input. Signal restoration from long wavelengths (low energy) to short wavelengths (high energy) requires additional energy. The pass gates can provide this functionality if the input to the pass gate is generated from an external far-field optical source, called an optical pump (conceptually analogous to Vdd), and the gate is controlled by the signal to either invert or pass. These restoring gates are shown in a darker shade in Figure 1. Since "Vdd" is a far field signal it does not need to be routed. This lack of routing overhead applies to all global inputs (like the clock signal) and is a significant advantage of RET-logic in area-constrained designs. From first-principles analysis we expect the switching time for a FO1 pass-gate to be approximately 2ns with a thermally dissipated power of less than 0.4nW. Communication with external receivers can be implemented using open-ended wires with dedicated emission wavelengths.

The memory cell in Figure 1 is a volatile SRAM design. Non-volatile memory could similarly be implemented using a special class of chromophores that have two switchable, stable molecular states [10]. The drawback of using non-volatile photo-switches is that the write time is much slower than that of an SRAM cell (ms vs. ns). We leave exploring non-volatile memory in nSPs as future work.

2.2 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 [14]. The fundamental property that these sensors employ is the dependence of RET on chromophore separation. RET only occurs when chromophores are sufficiently close (a few nm), and decreases as the distance increases. For sufficiently separated chromophores, RET will not occur. Sensors can be designed to enhance RET or prevent RET when a molecular recognition event occurs. Since RET sensors use the same inter-chromophore energy transfer as RET-logic they can interface directly with RET-logic circuits.

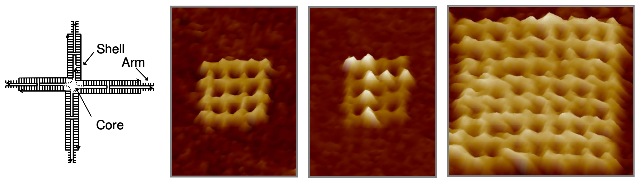
2.3 Preliminary Experimental Results

This section presents initial results toward fabricating an nSP using RET-logic. The purpose of this experiment is to demonstrate three main points: 1) that with the DNA grid we can place chromophores sufficiently close to achieve RET interaction, 2) demonstrate simple wired-OR RET logic, and 3) show that the DNA grid can bind analytes (in this case proteins) using antibodies placed at precise locations on the grid surface.

The cruciform motif we use to build the DNA grid is shown in Figure 2a. The motif is composed from three smaller motifs: a core, four shells, and four arms. Figure 2 also shows AFM images of (b) a 60x60nm hierarchically assembled grid, (c) the same grid with a protein pattern and (d) a 140x140nm grid each assembled in our laboratory using existing methods [18].

Using chemistry similar to that used to attach proteins it is possible to attach chromophores to specific sites on the DNA grid. The available sites on each grid occur at the intersection between motifs and at the center of the cruciform motif. The spacing between sites at the motif intersections is ~1.3nm and ~20nm between motif centers. The wired-OR gate is assembled using three chromophores, two for the input signals (Oregon Green and Alexa Fluor 535) and one for the output signal (Rhodamine Red) attached to the grid as shown in Figure 3. The two inputs are excited by wavelengths of light at 488nm and 518nm 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 590nm.

We experimentally assemble DNA substrates with the attached OR-gates as described above. A fluorometer measures the output of the assembly in the 300-800nm range under various input conditions. Input excitation is generated by a custom dual-beam excitation source. We estimate that



(a) (b) (c) (d)

Figure 2. (a) Schematic of a Cruciform Motif and Atomic Force Microscopy Images of (b) 60x60nm DNA Grid , (c) Grid with Bound Proteins and (d) 140x140nm Grid.

A) Pathogen Counting

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

B) Multi-Analyte Probe

```
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)
```

C) Finite Impulse Filter

```
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) }
```

D) Kinetic Analysis

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

E) Imaging

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

Table 1. Pseudocode for nSP target applications.

the sample contains ~10^{12} gates. The experimental results using inputs of 488nm (IN 1), 518nm (IN 2) and simultaneous 488nm/518nm (IN 1 + IN 2) are shown in the table in Figure 3. We isolate the specific contribution of the output (RR) chromophore due to RET from the background fluorescence by subtracting the normalized readout of a baseline grid assembly with the same chromophores placed at distances much greater than their respective near-field interaction radii (thereby preventing RET).

These results demonstrate the capability to place three chromophores sufficiently close to transfer excited-state energy from two distinct inputs to the same output, characteristic of an OR-gate. As part of our future work we are exploring the fabrication of inverting and non-inverting pass gates.

The technology presented in this section forms the foundation for biologically compatible computing and sensing, and we believe it will be possible in the future to fabricate nSPs. Understanding the technology is only part of architecting a system. We must also gain an understanding of the application requirements. The next section discusses several potential applications.

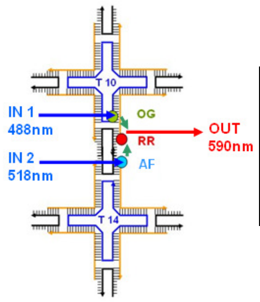
3. nSP Applications

One of the critical design requirements for an nSP is that it be small enough to diffuse through small volumes. A typical red blood cell width is approximately 6-8µm and we envision applications that require operation within a cell. The RET-logic nanotechnology described above only provides the potential to create appropriately sized nSPs. To meet the 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.

To illustrate the diverse application space we selected several target applications that we believe are representative of an nSP's potential. These applications are: 1) early disease detection, 2) custom multi-analyte molecular probes, 3) molecular kinetics analysis, 4) monitoring complex biological scale processes, and 5) imaging below the diffraction limit. We discuss each application in more detail below.

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: 1) the algorithm to perform and 2) timing constraints. The algorithm specifies, at a high level, the computation to perform and is used 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 timescales are in the range of seconds to minutes [9]. These times are generally determined 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 is bound forever. In contrast, for reversible sensors the analyte will eventually detach from the sensor. Timescales for irreversible sensors are determined by the binding rate of the analyte. A reversible sensor's time dependent behavior can be characterized by two parameters: 1) a period, and 2) a dwell time. The period is defined as 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.



Input 1 (488nm)	Input2 (518nm)	RET Output (590nm)
OFF	OFF	0
ON	OFF	39
OFF	ON	31
ON	ON	70

Figure 3. Observed RET Output (in Optical Fluorescence Counts) from the OR-gate with 488nm (IN 1) and 518nm (IN 2) excitation.

We present pseudocode for each of the applications 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 (i.e., read_sensor(name)) that targets a particular analyte and returns a Boolean value, True when the analyte is bound to the sensor, False otherwise. We also assume the ability to output an integer value.

Early disease detection. Integrating multiple pathogen sensors on an nSP can address the early detection problem through in vivo or in vitro biological monitoring over extended time windows. A program can count the number and type of binding (or detection) events over this window. At periodic intervals the nSP is queried for their event counts and the information is aggregated and compared against normal thresholds. The algorithm shown in Table 1A is an algorithm for counting pathogen binding events. An anti-hemagglutinin based sensor that detects the influenza A typical virus has a dwell time of 1-1.5 seconds and an expected period of minutes [15] at dilute concentration.

Custom Multi-Analyte Molecular Probes. Integrating several molecular probes on an nSP presents the possibility of detecting complex sequences and combinations of environment conditions. Table 1B shows an example program that outputs True when a specific combination and order of analytes is detected. The timescale is defined by the rates of monitored reactions and varies from minutes to ns depending on their biological or non-biological nature.

Kinetic Analysis. The quantitative evaluation of specific, reversible molecule binding is universally important in biology. Proteins interact with nucleic acids in gene expression, enzymes with substrates and inhibitors in metabolic processes, antigens with antibodies in the immune system. The binding interaction of two molecules at equilibrium is characterized by binding and dissociation constants. Classical methods for measuring binding constants with biosensors work at the macroscale and can involve many time consuming repetitive steps [20]. nSPs can derive sensor-analyte binding kinetics information in localized nanoscale environments in real-time by sweeping the nSP clock cycle time. The clock frequency is adjusted until the period and the dwell time of the analyte are captured in a sensor integration window. The algorithm integrates the instant sensor value for a fixed number of iterations and outputs the result (Table 1D). The range of available nSP clock frequencies determines the range of binding rate constants that can be analyzed. Classic macroscale methods, e.g., quantitative affinity chromatography, can measure binding constants ranging from 10^2–10^9 mol/L.

Monitoring Complex Processes. The ability of nSP chips to store and process nanoscale data across potentially large reaction time windows could be used, for example, to compute the average binding rate of a set of proteins or mRNA molecules. The challenge is to perform this averaging at the nanoscale over a large set of possible proteins (e.g., approximately $4x10^5$ unique proteins can be found in any individual human cell). A distributed set of nSPs, each designed to detect a subset of the proteins, could employ diffusion to sample and average protein concentrations over a large observation window to track protein expression.

In this type of complex process we expect nSPs to process long series of sensor data using accumulation, weighted averages, histograms or filters. Table 1C shows the algorithm for a Finite Impulse Response (FIR) filter. In the case of mRNA and proteins involved in gene activity the kinetics are on a timescale of seconds to minutes [9], although some processes, e.g., conformational changes of signaling proteins, are at the ms scale [25].

Functional Imaging below the Diffraction Limit. The sensor array of an nSP also provides implicit spatial information because of the precise, pre-determined location of each sensor. An interesting potential application of this fact is the imaging of features smaller than the diffraction limit, the fundamental resolution limit of optical microscopes. Multiple nSPs tiled on the surface of interest can serially transmit the 2D "image" of their sensing arrays and optical microscopy equipment can then combine the absolute nSP orientation data in the optical image (a feature that is above the diffraction limit) with the received nSP-relative information to create a composite image of all sensors across all nSPs. The same approach could potentially be used to implement high density nanoscale gene chips. The algorithm is shown in Table 1E, each nSP simply outputs the current values for its entire sensor array.

Other applications may be developed if nSPs become available, such as experiment-on-a-chip or nanoscale sensor networks. We leave further exploration of additional applications as future work, and instead focus on designing a single nSP based on our five representative applications.

Through inspection of the above applications we can extract several important characteristics that influence architectural designs, these include: 1) long time scales, 2) accumulating values, 3) waiting for an event, and 4) processing groups of individual sensor values as an aggregate. The following section discusses how these characteristics guide our architectural design.

4. An nSP Architecture

In this section we present our design for an nSP. We arrive at this design by combining the various application characteristics with the overarching requirement that an nSP must be small enough to diffuse through small volumes. We begin this section by discussing how these requirements qualitatively influence nSP architecture. This is followed by a detailed presentation of our nSP architecture. Our goal is to demonstrate that an nSP can be designed to meet the requirements of this new computing domain. We leave optimizing the architecture as future work.

4.1 Qualitative Architectural Implications

The nSP architecture is influenced by each of the application characteristics either individually or when combined with other characteristics. We discuss each of the characteristics and its qualitative influence on architecture. First, the long time scales (seconds to minutes) of biological applications implies that we do not need a high performance processor core (e.g., no superscalar, out-of-order, deep pipeline). Instead the architect can trade area for time using a very simple processor core with complex operations synthesized in software (e.g., multiply and divide).

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, the computations can generally be performed using fixed point arithmetic, avoiding the need for floating point hardware. Second, the accumulated value may require a larger range than the input value, thus leading to datapath components wider than the memory/sensor width. Third, accumulation is common enough that architectural support is justified to help reduce code size.

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

Finally, processing individual sensors as a group implies that it may not be necessary for the architecture to support access to individual sensor values. Instead it may be beneficial to provide support for processing multiple sensor values as a single entity.

4.2 nSP Overview

We investigated 8-bit architectures for ultra-small controllers, e.g., the ST Microelectronics ST6 [21] and the Freescale RSO8 [8], designed to be efficient and cost effective with small memory sizes. These architectures, although simple compared to mainstream designs, proved too complex for the extreme area constraints of nSPs. We instead elected to create an architecture streamlined for the expected applications and hardware limitations of nSPs.

Our nSP architecture is a simple accumulator-based processor with a small amount of addressable memory/sensors. Using a single-accumulator reduces the processor core complexity and enables 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/sensor memory space. Instructions are variable-length (4bit or 12bit) to decrease the memory footprint of application code. A special variant of our architecture, called Tiny, is designed for the smallest nSPs with a total memory space of only 8 bytes (16x4-bit).

4.3 Integrated Sensing

The sensor-centric nature of nSP applications means that the interface between processing and sensing plays an important role in the system design. There are a variety of ways that an nSP can support sensing. For instance, sensors could be memory mapped. In this scenario, predetermined memory addresses are set aside for access to sensed values via load instructions. The method we explore in this paper exploits the biological compatibility of the entire system when using technologies like RET-logic. Since RET is the method used for sensing and for computing, we can directly integrate the sensing mechanisms into the system design.

Although there may be many ways to exploit an integrated design methodology, in this paper we examine methods for sensing to directly modify memory locations. Specifically, certain SRAM cells can be augmented with appropriate sensing mechanisms that force the memory location to the value “1” (or “0” as needed). A technological requirement is that the active area of the sensor must fit within the confines of an SRAM cell. These environmentally modified memory locations could be designed to provide a set of memory-mapped sensors. However, this does not exploit the full potential of a unified sensing and logic technology such as RET. Instead we can interleave sensor-augmented SRAM cells with standard SRAM cells, including those used for instruction opcode bits. For example, a JMP opcode could be modified into a NOP opcode by a sensor binding event. Similarly, instruction operands could be modified by 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/software co-design 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, a single branch instruction could be used that changes to a nop (and escape) when the protein binds.

Using IFS can dramatically improve code density where it is applicable. There is, however, a trade-off: the instruction becomes statically bound to the sensor. This can prevent code reuse via procedures or looping. We note that code reuse is valuable when the increase in code-size due to procedure overhead is compensated by the removal of significant inline code. In the limited memory and program space of an nSP this break-even can be difficult to reach. We evaluate this trade-off in more detail in Section 5.

4.4 The nSP ISA

Table 2 shows our nSP ISA which supports several common memory, arithmetic, and control transfer instructions. Each of the instructions in the ISA is included because it directly supports common operations in one or more applications. Note that the ISA lacks several instructions one often considers standard, e.g., subtract, multiply, and divide. These operations can all be synthesized with the provided instructions if needed. For brevity, when details are necessary we discuss only the Standard nSP ISA.

Several of our nSP instructions are designed to reduce application code size. The 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, BNZ, INC and INCI all have the option of using IFS. When declared sensitive to an analyte (A), these instructions turn into NOPs when that analyte binds to the appropriate memory location. To facilitate this mechanism we encode a single-bit difference between the opcode of these instructions and the NOP opcode. When a sensor is fused to this bit location the opcode value depends on the value of the sensor, as determined by the presence or absence of an analyte. This requires careful hardware/software co-design to ensure proper alignment of code and sensors in the memory space.

We denote the analyte-dependence of an instruction by placing the analyte name in parenthesis after the instruction. Depending on the encoding of the binding event, the instruction can be NOP-ed by either the presence or the absence of the analyte. We use C-style (!) to indicate the instruction is executed when the analyte is not sensed and the analyte identifier alone when the instruction is executed only in the presence of the analyte. For example, INC (!A) means the accumulator is incremented only if the analyte A is not present, else it is a NOP and has no effect.

Instruction operands, either immediate values or addresses, are 8-bit length, the same as the data granularity of

Instruction	Op-code	Standard RTL	Tiny RTL	Description
LD addr	0000	ACC[0..7] = M_8[addr]; PC += 3	ACC[0..3] = M_4[addr]; PC += 2	Load from memory
ST addr	0001	M_8[addr] = ACC[0..7]; PC += 3	M_8[addr] = ACC[0..4]; PC += 2	Store to memory
ADD addr	0010	ACC[0..15] += M_8[addr]; PC += 3	ACC[0..7] += M_4[addr]; PC += 2	Add unsigned from memory
ADDI imm	1100	ACC[0..15] += M_8[PC+1]; PC += 3	ACC[0..7] += M_4[PC+1]; PC += 2	Add unsigned immediate value
SHL	0100	ACC[0..15] = ACC[1..15] | 0; PC += 1	ACC[0..7] = ACC[1..7] | 0; PC += 1	Shift 1 position left
SHR	0101	ACC[0..15] = 0 | ACC[0..14] ; PC += 1	ACC[0..7] = 0 | ACC[0..6] ; PC += 1	Shift 1 position right
NOT	0110	ACC[0..7] = ! ACC[0..7]; PC += 1	ACC[0..4] = ! ACC[0..4]; PC += 1	Bitwise NOT
AND imm	1010	ACC[0..7] += M_8[PC+1]; PC += 3	ACC[0..4] += M_4[PC+1]; PC += 2	Bitwise AND
CLR	0011	ACC[0..15] = 0; PC += 1	ACC[0..7] = 0; PC += 1	Clear ACC
JMP addr (A)	1110	PC = M_8[PC+1]	PC = M_4[PC+1]	Jump to address
BNZ addr (A)	1101	If ACC[0..7] != 0 PC = M_8[PC+1] Else PC += 3	If ACC[0..4] != 0 PC = M_4[PC+1] Else PC += 2	Jump to addr if ! ACC
INC (A)	1011	ACC[0..15] += 1; PC+=1	ACC[0..7] += 1; PC+=1	Increment ACC

INCI imm (A)	0111	ACC[0..7] =M_8[PC+1];
		ACC[0..15]+=1;
		M_8[PC+1]=ACC; PC+=3
			ACC[0..3] = M_4 [PC+1];
			ACC[0..7]+=1;
			M_4 [PC+1]=ACC;
			PC+=2
				Increment memory value, store in memory and ACC

NOP	1111	PC += 3	PC += 2	No operation
OUT addr	1000	COMM = M_8[PC]; PC += 3	COMM = M_4[PC]; PC += 2	Output ACC
OUTCLR addr	1001	COMM = M_8[PC]; PC +=3	COMM = M_4[PC]; PC +=2	Output & clear memory

Table 2. Our nSP ISA: M_8 = 8 bits starting at specified location, M_4 = 4 bits. Standard has 16-bit Accumulator, 8-bit PC, and 255 4-bit memory locations. Tiny has 8-bit Accumulator, 4-bit PC, and 16 4-bit memory locations.

loads and stores. This allows for easy address manipulation, but most importantly it increases the numerical range for operations performed on sensor data. One of the common patterns we see in the nSP target applications is that individual sensor data is accumulated in 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 would result from emulation with 4-bit instructions. The accumulator is 16-bit for extended dynamic range in applications like FIR filters where double word samples are averaged or multiplied through shift-and-add. The high 8 bits of the accumulator can be accessed through explicit shifts into the low 8 bits.

Defining only 8-bit wide memory operations could prove restrictive given the single-bit granularity of sensors. Indeed, accessing an individual sensor value through a wide load can require additional bit masking and shifting. We argue, however, that in cases where single bit sensor values are necessary it is generally more efficient to use the compact instruction-fused sensing mechanism. An analysis of the target application algorithms shows three distinct cases in which sensors are individually queried for their value: 1) branch outcome depends on an individual sensor value, 2) arithmetic result depends on an individual sensor value, 3) output of sensor values. The first two can be addressed with IFS and the latter can bundle values for output and separate values at a remote receiver.

4.5 Discussion

The above nSP architecture is designed to meet the requirements of the new computing domain of biological scale computing. The architecture is generic and can be implemented 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. The architecture presented in this section provides a set of novel mechanisms (e.g., instruction-fused sensing) that enable compact implementations of important biological applications. The following section analyzes the ability of the proposed architecture to satisfy the requirements of the target applications.

5. Evaluation

In this section we evaluate our nSP architectures. We begin with an analysis of the nSP size, followed by a discussion of the applications implemented using the nSP ISA and the memory resources required for execution. We then explore the impact of instruction-fused sensing on program size and conclude with simulation of applications monitoring time varying environmental conditions.

5.1 Node Size

As previously noted, the size of an nSP is an important factor for biological applications. Here we use the RET-logic layouts and characteristics described in Section 2.1 to estimate the area of nSP implementations. Our preliminary layout indicates that the Standard nSP implementation with 128 bytes of memory requires 2.5 x 2.5μm and a Tiny nSP implementation with 8 bytes of memory requires 800x800nm. With a surface area comparable to the largest known virus [22], the Tiny nSP could diffuse in the same environments as the virus (e.g., a cell).

5.2 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 between 9 and 11.5 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 is dependent on the number of monitored sensors; more sensors requires more memory space. For brevity, we present the Standard nSP code for only two applications, pathogen counting and a multi-analyte probe.

The nSP program in Figure 4 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. An 8-bit counter is used 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. This compact code is achieved by using IFS for event detection (i.e., 1) a pathogen is present and 2) the result should be transmitted). The INCI instruction also provides compact accumulation of the counter.

```
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 4. 8-bit event counter (Pathogen Counting).

```
0  JMP (!A) 0  // if not A jump to self
3  JMP (!B) 3  // if not B jump to self
6  JMP (!C) 3  // if not C jump to previous
9  JMP (!D) 9  // if not D jump to self
12 JMP (E)  9  // if not E jump to done
15 OUT 1       // output non-zero value
```

Figure 5. Multi-analyte probe implementation.

The role of a multi-analyte probe is to detect specific sequences of analytes and their time order, in this example (A) then (B and C) then (D and not E). We use IFS to map the analyte ordering on top of executed control logic (Figure 5) and guarantee that the program control path, which is dynamically changed by the analytes, reaches its output phase only when it interacts with the target sequence of analyte events. A single IFS instruction (JMP) is sufficient to sense and process each analyte in the intended evaluation sequence. The total memory footprint for the Standard nSP implementation is 8.5 Bytes.

The remaining applications all require less than the 128 Bytes available on the Standard nSP. Our FIR implementation, that uses a moving average, is the largest, most complex application and it requires just below 60 Bytes. Kinetic analysis and Imaging each require around 10 Bytes.

Table 3 shows the Tiny nSP implementations for four applications. FIR is too complex to fit within the 8 Byte limitation. The main cost of using Tiny nSPs is that dynamic range for counting and accumulation is reduced from 8 bits to 4 bits and the total number of sensors that can be monitored is much more limited. The fundamental advantage of the Tiny nSP is their diminutive size: they can diffuse through and sample molecular environments that are inaccessible to the larger Standard nSP.

5.3 Impact of Instruction-Fused Sensing

A single IFS instruction can replace several non-IFS operations and significantly reduce the memory requirement for applications. Figure 6 shows the IFS and non-IFS implementations of the sample multi-analyte probe application for the Standard nSP. The non-IFS code allocates memory locations for sensors and reads the sensor value with explicit load operations, leading to memory footprint of 21 Bytes versus the 9 Bytes of the IFS version.

Figure 7 shows the total memory required by IFS relative to non-IFS (load/store) implementations of our applications for the Standard nSP. IFS reduces the footprint between 58%, in the case of the multi-sensor analyte probe, and 5%, for the single-sensor, processing intensive FIR.

A tradeoff in using IFS is that instructions become statically bound to sensors, preventing conventional code size reducing techniques, like procedures or loops. Figure 8 explores this tradeoff. For each application we show on the y axis the number of sensors that can be processed by IFS and LD/ST implementations as we increase the maximum memory available on a Standard nSP. For both implementations we unroll loops when unrolling allows an increase in the number of sensors for the given memory size.

Within the range of our Standard nSP addressable memory, 128 Bytes, IFS is generally more efficient than LD/ST,

Counter	**Multi-Analyte**
`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	**Kinetic Analysis**
`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`

Table 3. Tiny nSP Programs (8 Byte Memory Space).

i.e., more sensors can be processed for a given memory size. An exception is the pathogen counter application where the traditional LD/ST implementation is more efficient when available memory is more than 64 Bytes and we wish to count more than 5 pathogens. With more memory, loop overhead can be amortized over more sensors. The same characteristic can be seen for the kinetic analysis application; however, in this case the break-even point where LD/ST becomes more memory efficient than IFS is beyond the maximum addressable memory range of our nSPs.

```
0  JMP (!A) 0
3  JMP (!B) 3
6  JMP (!C) 3
9  JMP (!D) 9
12 JMP (E)  9
15 OUT 3
```

```
0  LD 37
3  NOT
4  BNZ 0
7  LD 38
10 NOT
11 BNZ 7
14 LD 39
17 NOT
18 BNZ 7
21 LD 40
24 NOT
25 BNZ 21
28 LD 41
31 BNZ 21
34 OUT 3
37 sensorA
38 sensorB
39 sensorC
40 sensorD
41 sensorE
```

Figure 6. Code size impact of IFS on the multi-analyte probe.

Up: IFS (9 bytes)
Right: LD/ST (21 bytes)

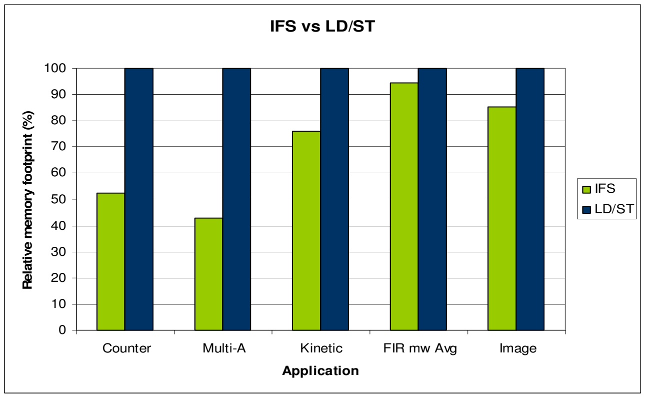


Figure 7. Impact of using IFS on the memory requirement of target nSP applications.

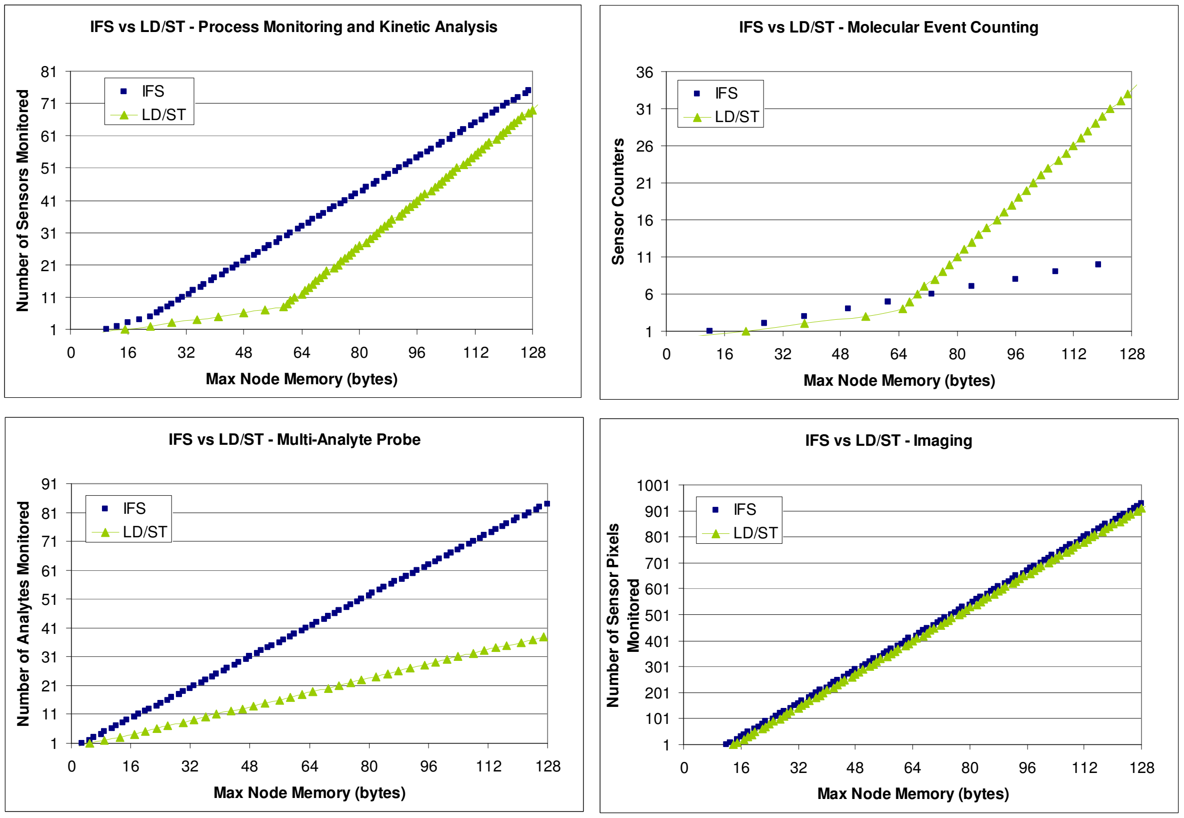


Figure 8. Comparison of IFS and LD/ST implementations of applications. The available nSP node memory (X axis) and the number of sensors to be processed (Y axis) influence the relative benefit of using IFS.

5.4 Application Simulation Results

We verify the expected output of nSP applications using a simple cycle-level nSP functional simulator augmented with chemical environment and sensor-analyte interaction simulation. Memory access (for a 4-bit word, including sensor read) and arithmetic operations are performed in a single cycle. The total number cycles per instruction varies between 2 and 6 depending on 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. The binding event probability is modulated by binding and dissociative rate constants which are explicitly specified for all distinct sensor-analyte pairs.

For each application we initialize the nSP with the appropriate program, simulate a chemical environment characteristic for that application and follow the program output in time. The following details the results.

Pathogen Counting. We simulate an influenza virus environment using published pathogen period and dwell time for anti-hemagglutinin based sensors [15]. Figure 9 (bottom graph) shows the output of a Standard nSP running the pathogen counting code from Figure 4 in the presence of a time-varying pathogen concentration (top). The nSP clock cycle time is 100Hz, and the output is requested, via an external optical send signal, every 1000 seconds.

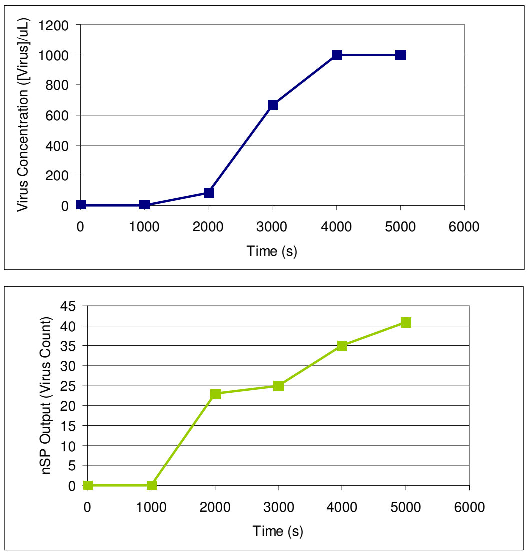


Figure 9. Influenza-A virus counting simulation. The nSP node output (bottom) follows the virus presence (top).

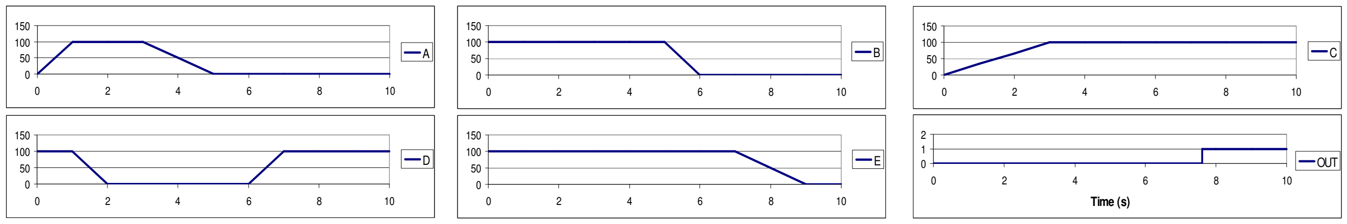


Figure 10. Multi-analyte probe simulation: the node output (OUT) depends on the sequence of analytes (A,B,C,D,E).

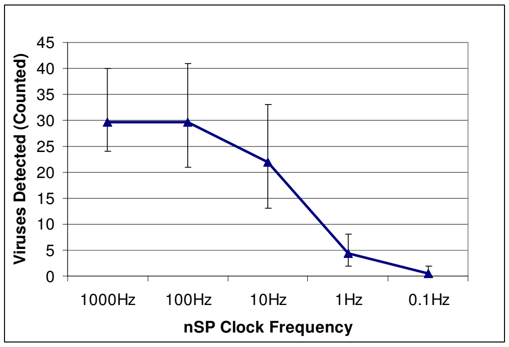


Figure 11. The pathogen binding kinetics determine the minimum nSP clockrate before counting becomes inaccurate due to missed events.

When requested, the program outputs the number of observed pathogen binding events since the last transmission. As expected, the value of the output is correlated with the pathogen concentration and shows variations due to stochastic (single-molecule) sensor binding. Observing this trend could be used to diagnose an infection.

The minimum nSP clock cycle time is determined by the timescale of the analyte-sensor interaction. If the program samples sensors too slowly, then binding events could be missed. In this experiment the critical sampling threshold is less than 1 second (the average dwell time for this pathogen-sensor combination). Thefore, the nsP clock rate must be fast enough to ensure the program samples the sensors frequently enough. Figure 11 shows the output of the virus counting application running at various nSP clock rates (averaged over ten queries taken 1000 seconds apart). The virus concentration and the window of time over which the nSP counts binding events are held constant. Given our multicycle nSP, the clock frequency must be 100 Hz or greater to avoid a misdiagnosis. At lower clock rates pathogen binding events are missed as reflected by the decreasing counter value.

Multi-Analyte Probe. We model 5 generic analytes (A-E) and execute the molecular probe application code from Figure 5, which detects the (A) then (B and C) then (D and not E) sequence of events. Figure 10 shows the input concentration of each analyte and the program output (OUT) over time.

The result emphasizes the local, single-molecule sensing characteristics of the probe. Even though there is still some concentration of analyte E present in the system, the program asserts its detection output (just before the 8s mark). The reason is the stochastic nature of the output decision, 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.

6. 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 range. 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 early microprocessor designs [7] and the broad area of sensor networks [11, 13] and ultra-low power sensor processors [26]; however, our small scale creates significantly different resource constraints. Other work explores novel molecular logic, but currently only presents individual gate functionality [3].

Other closely related work includes the Decoupled Array Multiprocessor (DAMP) [4], the Nano-scale Active Network Architecture (NANA) [16] and the Self-Organizing SIMD Architecture (SOSA) [17] which all use DNA-based self-assembly of nano-electronic devices. Our work also uses DNA self-assembly but focuses on novel single molecule optical devices, thus achieves higher density and provides an efficient method for macro-scale interfacing (pitch matching in electronics).

7. Conclusion

Two driving forces on computer architecture are application requirements and technology change. The combination of important problems in the life sciences and advances in material science are exposing a new computational domain: biological scale integrated sensing and processing. The ability to utilize programmable devices at biological scales may enable life scientists to perform hypothesis testing previously thought impossible. This domain presents new challenges to computer architects due to the extreme size constraints: a device must be capable of diffusing through small volumes while still meeting application requirements.

This paper introduces an architecture for nanoscale sensing and processing. We analyze the application characteristics (e.g., long time scales and common operations) to design a multicycle accumulator-based architecture. A novel aspect of this architecture is the use of instruction-fused sensing that exploits the unified use of nanoscale devices for both sensing and logic design to allow sensors to directly modify logic values (i.e., instruction opcode bits). We implement several representative applications that execute on our proposed architecture and demonstrate capabilities (e.g., sensing based on complex logic) beyond those achievable with current simple biological sensors. The work presented in this paper represents our first steps toward developing biological scale computing systems.

Acknowledgments

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

References

[1] H. Abelson, et al., "Amorphous Computing," *Communications of the ACM*, vol. 43 (5), pp. 74-82, 2000.

[2] A. Bachtold, et al., "Logic Circuits with Carbon Nanotube Transistors," *Science*, vol. 294, pp. 1317-1320, 2001.

[3] A. P. de Silva and N. D. McClenaghan, "Molecular-Scale Logic Gates," *Chemistry - A European Journal*, vol. 10 (3), pp. 574-586, 2004.

[4] C. Dwyer, et al., "DNA Self-assembled Parallel Computer Architectures," *Nanotechnology*, vol. 15, pp. 1688-1694, 2004.

[5] C. Dwyer, et al., "Energy Transfer Logic on DNA Nanostructures: Enabling Molecular-Scale Amorphous Computing," in *Proceedings of the 4th Workshop on Non-Silicon Computing (NSC4)*, pp. 33-40, 2007.

[6] D. Endy, "Foundations for Engineering Biology," *Nature*, vol. 438 (7067), pp. 449-453, 2005.

[7] F. Faggin and M. E. Hoff, "Standard parts and custom design merge in four-chip processor kit," *Electronics*, pp. 112-116, 1972.

[8] S. Freescale, "MC9RS08KA2, MC9RS08KA2 Datasheet," 2007.

[9] I. Golding, et al., "Real-Time Kinetics of Gene Activity in Individual Bacteria," *Cell*, vol. 123 (6), pp. 1025-1036, 2005.

[10] M. Heilemann, et al., "Carbocyanine Dyes As Efficient Reversible Single-molecule Optical Switch," *Journal of the American Chemical Society*, vol. 127 (11), pp. 3801--3806, 2005.

[11] J. Hill, et al., "System Architecture Directions for Networked Sensors," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 93--104, 2000.

[12] Y. Huang, et al., "Logic Gates and Computation from Assembled Nanowire Building Blocks," *Science*, vol. 294, pp. 1313-1317, 2001.

[13] P. Juang, et al., "Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with Zebranet," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 96--107, 2002.

[14] J. R. Lakowicz, *Principles of Fluorescence Spectroscopy*. New York: Kluwer Academic / Plenum Publishers, 1999.

[15] F. Patolsky, et al., "Electrical Detection of Single Viruses," *Proceedings of the National Academy of Sciences*, vol. 101 (39), pp. 14017--14022, 2004.

[16] J. P. Patwardhan, et al., "NANA: a Nano-scale Active Network Architecture," *J. Emerg. Technol. Comput. Syst.*, vol. 2 (1), pp. 1-30, 2006.

[17] J. P. Patwardhan, et al., "A Defect Tolerant Self-organizing Nanoscale SIMD Architecture," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 241-251, 2006.

[18] C. Pistol and C. Dwyer, "Scalable, low-cost, hierarchical assembly of programmable DNA nanostructures," *Nanotechnology*, vol. 18, pp. 125305-9, 2007.

[19] C. Pistol, et al., "Nanoscale Optical Computing using Resonance Energy Transfer Logic," *IEEE Micro*, vol 28 (6) Nov-Dec, 2008.

[20] R. L. Rich and D. G. Myszka, "Survey of the year 2001 commercial optical biosensor literature," *Journal of Molecular Recognition*, vol. 15 (6), pp. 352-376, 2002.

[21] Stmicroelectronics, "ST6200C/ST6201C/ST6203C Datasheet," 2007.

[22] M. Suzan-Monti, et al., "Genomic and evolutionary aspects of Mimivirus," *Virus Research*, vol. 1 (117), pp. 145-155, 2006.

[23] C. M. Tsai and C. E. Frasch, "A sensitive silver stain for detecting lipopolysaccharides in polyacrylamide gels," *Anal Biochem*, vol. 119 (1), pp. 115-119, 1982.

[24] T. Vo-Dinh, et al., "DNA Biochip Using a Phototransistor Integrated Circuit," *Analytical Chemistry*, vol. 71 (2), pp. 358--363, 1999.

[25] A. J. Wand, "Dynamic activation of protein function: A view emerging from NMR spectroscopy," *Nature Structural Biology*, vol. 8 (11), pp. 926-931, 2001.

[26] B. Zhai, et al., "A 2.60pj/inst Subthreshold Sensor Processor for Optimal Energy Efficiency," *VLSI Circuits, 2006. Digest of Technical Papers. 2006 Symposium on*, pp. 154--155, 2006.

[27] T. Zhang, et al., "Recent Progress in Carbon Nanotube-based Gas Sensors," *Nanotechnology*, vol. 19 (33), 2008.