Lax: Driver Interfaces for Approximate Sensor Device Access

Phillip Stanley-Marbell   Martin Rinard
MIT
psm@mit.edu, rinard@csail.mit.edu

Abstract

Embedded sensor platforms can dissipate most of their energy in accessing sensor integrated circuits such as gyroscopes. But the algorithms which process the sensor data and the humans who consume the overall output of the system may often be able to tolerate some amount of error in the retrieved sensor values. Because devices are accessed through interfaces provided by system software, exploiting the tolerable error for improvements in energy efficiency requires appropriate system software and hardware support. However, no such support currently exists.

We present Lax, a device driver abstraction for interacting with sensors that enables power savings in exchange for occasionally returning erroneous sensor data. Our implementation on a hardware prototype delivers savings in sensor dynamic power dissipation of up to 48% (as compared to precise device access) while providing sensor access error rates lower than 5 data acquisition errors per 100 data accesses. Given the significant proportion of system energy budgets in wearable platforms that are devoted to sensors, approximate sensor data acquisition using Lax can deliver significant system-level energy savings.

1. Introduction

Research on energy efficiency in microsensor platforms has focused, among other things, on improving the efficiency of computation and communication, and meeting the unique task scheduling and resource availability fluctuation challenges of these platforms [11, 15, 28]. In state-of-the-art platforms however, the power dissipation in accessing sensors may equal or surpass that of the compute elements (Figure 1). The received wisdom holds that reducing the power dissipation of sensor integrated circuits such as gyroscopes and temperature sensors requires circuit and fundamental semiconductor technology advances. We advocate a new and complementary power-reduction approach which builds on insights into the role of sensors in systems.

1.1 Focus on the true bottleneck: sensor power

Sensors are becoming the dominant source of power dissipation in environment monitoring, wearable, and health-tracking systems. Several characteristics make sensors especially amenable to energy savings through approximation.

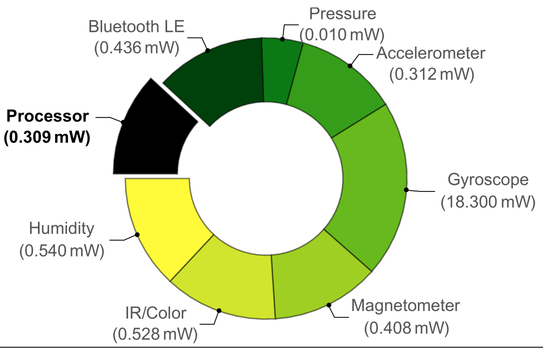


Figure 1. Logarithmically-scaled sector plot of relative power dissipation while active, for several state-of-the-art sensors [3, 19, 20, 23, 24], a Bluetooth Low Energy radio [22] in *advertising/discoverable* mode, and an implementation of the lowest-power ARM architecture variant currently available (ARM Cortex M0+ [8]) running a `while(1)` loop from its on-chip SRAM at 2 MHz and 3.0 V. All but one of the sensors use more power than the processor.

First, sensors are typically *stateless*, so that errors in one sample do not propagate to affect the next sample. Digital computation, in contrast, is typically stateful, so that errors in control logic may accumulate over time. Second, sensor interaction is typically *transactional*, with the obtained values having limited temporal influence. Third, because sensors are primarily *analog* circuits, they are better able to tolerate approximate computing mechanisms that can lead to catastrophic failures when applied to digital microprocessors [12].

But sensor approximation must be appropriately controllable to be successful. Algorithms that consume sensor output must be able to specify when and how much approximation they can tolerate and the low-level sensor interface software must deliver approximate sensor accesses that satisfy these specifications.

1.2 Lax

As one approach to this challenge, we are developing Lax, a device driver abstraction and associated hardware support, that exploits opportunities to trade sensor accuracy for significant energy savings. Lax is based on the insight that in certain phases of their lifetime, the algorithms which process sensor data, or the humans who consume the overall output of embedded sensor systems, may be able to tolerate some amount of error in retrieved sensor values. Lax provides a device driver abstraction that enables sensor data consumers to specify how much sensor error they can tolerate. Armed with this specification of acceptable error levels, Lax then controls the sensor's electrical interface to minimize sensor energy consumption while still delivering acceptable sensor accuracy. Our proof-of-concept implementation realizes this tradeoff via a combination of software and printed circuit board components (there are no required changes to the sensor integrated circuits).

To demonstrate the feasibility of Lax, we performed sensor output data collection and power measurements on a setup comprising a state-of-the-art ARM Cortex-M0+ processor and sensors typical of contemporary sensing and wearable platforms. The setup uses two programmable voltage regulators to control the supply and I/O interface voltages of sensors across nine discrete voltage levels. To show that the circuit-level changes needed to realize our prototype setup in deployed systems can be implemented with low overhead, we have designed (but not yet manufactured) a custom printed circuit board implementation of a demonstrator system. This implementation shows how an entire system of sensors, processor, radio interface, and all the necessary components to implement our proposed techniques, can be implemented in ~625 mm^2, with the additions required to implement Lax occupying only ~15 mm^2.

1.3 How Lax works

Lax uses two techniques, a combination of software and minimal hardware support, to trade efficiency for accuracy:

❶ **Device abstractions for approximation.** Lax provides a device driver abstraction, detailed in Section 2, through which it enables sensor device accessors to specify the tolerable degree of imprecision and unreliability. Some sensors [9] already have limited support for modes which trade resolution for access power; for these, Lax can exploit the extant hardware facilities.

❷ **Sensor supply scaling.** Regardless of already existent hardware support for trading precision, accuracy, or reliability for power consumption, many sensors can be operated outside their specified supply voltages. This enables usable tradeoffs between the reliability of data acquisition, fidelity of data provided, and power dissipation. We detail our implementation of this hardware support in Section 3.

1.4 The prospects for Lax

To illustrate the potential power dissipation versus error tradeoffs of Lax under sensor supply scaling, Figure 2 plots power dissipation and retrieved sample noise properties as a function of supply voltage for two state-of-the-art sensors. For both the humidity sensor and accelerometer, power dissipation decreases by a factor of ~4× with a halving of supply voltage. For the accelerometer, the noise in the retrieved signal (measured, by convention, in micro-gravities per square-root-Hertz, $\mu\text{g}/\sqrt{\text{Hz}}$) increases by a factor of ~1.5× with a halving of the supply voltage [1], as shown inset in the figure.

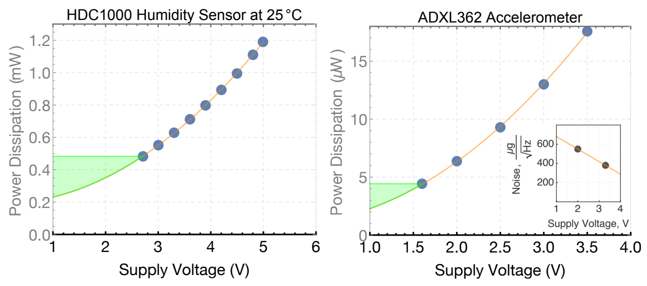


Figure 2. Manufacturer-reported power dissipation [1, 23] for reliable operation at recommended supply voltages (points), extrapolated to lower voltages using a physics-based model (line). It falls by $60.6(1-k^2)$ and $62.6(1-k^2)$ percent respectively (shaded region), for each factor-k reduction from the lowest voltage for reliable operation.

These savings are larger than the hypothetical savings posited for future *approximate processors* [7, 10] and can be achieved without resorting to the complex microarchitectural changes required by proposed mixed-accuracy processors. Our preliminary measurements, presented in Section 3, agree with these illustrative extrapolations, and show power savings of up to 48% while incurring fewer than 5 sensor access errors per 100 accesses.

1.5 Example: Lax in Systems

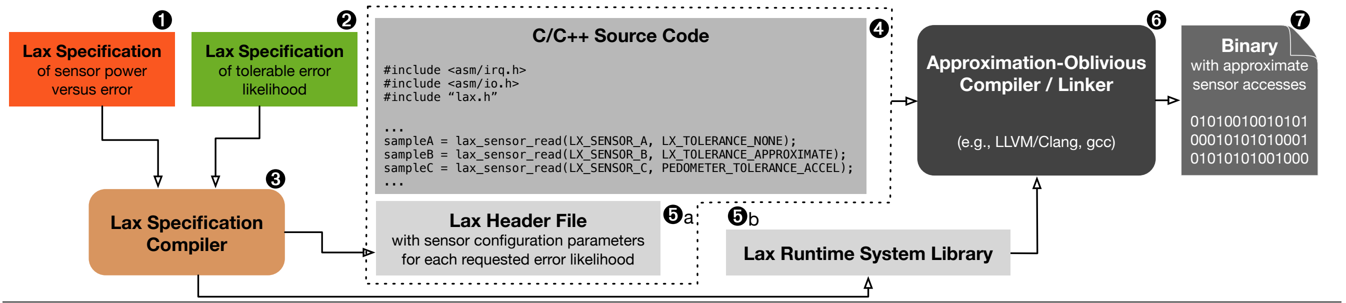
Using contemporary operating systems, applications cannot specify how much precision, accuracy, or reliability they require from a sensor.

Figure 3 shows the block diagram for a pedometer application, as might be incorporated into popular wearable health-tracking platforms. The figure shows the data flow from an accelerometer sensor through blocks of the signal processing needed to perform step counting [29]. All the facilities for Lax can be logically interposed in the interface to data acquisition, as shown in the figure.

2. The Lax Device Interface

System software interfaces for approximate device access should enable the specification of three types of tolerance to deviations from correct behavior:

- **Latency tolerances.** Different applications may be able to tolerate differing latencies in retrieving values from a sensor. This can be exploited by the hardware facilities described in Section 3 to reduce the energy required per sensor sample acquisition.



```
#include <asm/irq.h>
#include <asm/io.h>
#include "lax.h"

...
sampleA = lax_sensor_read(LX_SENSOR_A, LX_TOLERANCE_NONE);
sampleB = lax_sensor_read(LX_SENSOR_B, LX_TOLERANCE_APPROXIMATE);
sampleC = lax_sensor_read(LX_SENSOR_C, PEDOMETER_TOLERANCE_ACCEL);
...
```

Figure 4. In this example, software uses Lax primitives to request sensor values (block ❹). The amount of inaccuracy, imprecision, or unreliability that is tolerable in responses to those requests is either specified using defaults such as **`LX_TOLERANCE_NONE`**, or application-specific tolerances such as **`PEDOMETER_TOLERANCE_ACCEL`**. Although not mandatory, if used, the meaning of these optional constants are specified explicitly in the tolerable error specification (block ❷). The Lax specification compiler must combine these with the hardware error characteristics (block ❶) to emit source and headers that implement the approximate sensor access (block ❺).

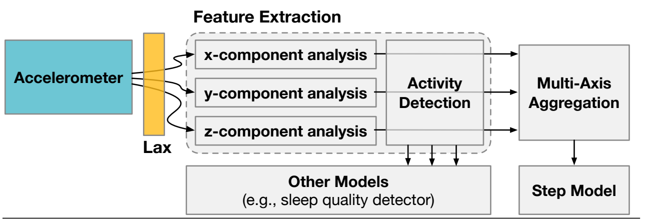


Figure 3. Data flow in a pedometer application. Samples from the x-, y-, and z-components of acceleration are typically first low-pass filtered, then fed into an activity detection algorithm. If the signature matches walking, these acceleration components are fed into a model for predicting steps from acceleration signatures.

- **Loss or throughput tolerances.** When the algorithms consuming sensor data can tolerate occasional wholly-incorrect or missing samples, knowledge of this tolerance of unreliability can be used to reduce sample acquisition energy.
- **Value deviation tolerances.** When the algorithms consuming sensor values can tolerate small deviations from accuracy or precision in sensor readings, this can yet again be exploited to reduce sample acquisition energy.

Lax enables driver writers to specify tolerances to these types of behavioral aberrations. These specifications can then be exploited in existing system architectures, as illustrated in Figure 4.

2.1 Slax: Specifying Lax sensor access

Tolerances should be specified in the context of a given sensor type and should be statically checked because it is possible to specify meaningless, unattainable, or mutually-contradictory tolerance specifications. Lax provides a small domain-specific language, *Slax*, for defining tolerances. Slax captures the latency, loss, and value-deviation tolerances of sensor data acquisition and is thus complementary to interface definition languages such as Devil [16], which are intended to ease the construction of complete device drivers.

The grammar for Slax is shown in Figure 5, and an example specification for an accelerometer is given in Figure 6. A Slax specification comprises one or more **`sensor`** or **`tolerance`** blocks. The **`sensor`** blocks describe the error properties of sensors at various operating points, while the **`tolerance`** blocks denote groups of error tolerance settings that are required together at various points in an application.

In practice, a driver may use a Lax-default or driver-specified tolerance specification in accessing a given sensor, as illustrated in block ❹ of Figure 4. For example, given the Slax specifications in Figure 6, the following C fragment would employ the configuration implied by the constants **`PLATFORM_ACCEL_A`** and **`PEDOMETER_TOLERANCE_ACCEL`**:

```
/*   Use Lax to achieve lowest power for required accuracy.  */
sampleC = lax_sensor_read(PLATFORM_ACCEL_A,
                          PEDOMETER_TOLERANCE_ACCEL);
```

The Lax runtime must use the provided tolerance indicator to determine the best device operating point. When integrated into contemporary operating systems, it would then set the properties of the device using, e.g., **`ioctl()`** or equivalent system calls (our proof-of-concept implementation presented in Section 3 runs over bare metal). The **`sensor`** blocks on the other hand must be based on hardware characterizations. They would ideally be provided by a hardware platform designer or vendor, but could be overridden by a driver writer's own **`sensor`** block. We provide examples of the necessary characterizations that yield **`sensor`** blocks in Section 3.

2.2 Challenges

Even though the potential benefits of exploiting tolerance to imprecision, inaccuracy, and unreliability are significant, there are several challenges to implementing a system that can effectively trade those tolerances for performance or energy-efficiency. For example, a simplistic solution to determining a valid operating point from the **`sensor`** block for a given sensor might be straightforward, using, e.g., a

```
unsignedImm    ::= "0" | "1..9" {"0..9"} .
stringConst    ::= "\"" {Unicode Character} "\"" .
integerConst   ::= ["+" | "-"] unsignedImm .
dRealConst     ::= ("0" | "1..9" {"0..9"}) "." "0..9" {"0..9"} .
eRealConst     ::= (dRealConst | integerConst) ("e" | "E") integerConst .
realConst      ::= dRealConst | eRealConst .
rationalConst  ::= integerConst "/" integerConst .
numConst       ::= integerConst | rationalConst | realConst .

slaxSpec       ::= specHead {defn} .
specHead       ::= "specification" ident ";" .
ident          ::= {Unicode Character} .
defn           ::= sensorDefn | toleranceDefn .
sensorDefn     ::= "sensor" ident ["@"numConst units] "=" "{" {sensorStmt} "}".
toleranceDefn  ::= "tolerance" ident "=" "{" {toleranceStmt} "}" .
sensorStmt     ::= "provide" "(" eClass ")" "=" "{" cStmt {";" cStmt} "}" .
toleranceStmt  ::= "require" "(" eClass ")" "=" "{" cStmt {";" cStmt} "}" .
eClass         ::= "deviation" | "latency" | "loss" | "throughput" .
cStmt          ::= cmpOp numConst units ":" likelihoodExpr | alwaysExpr .
likelihoodExpr ::= "likelihood" cmpOp numConst "in" numConst "readings" .
alwaysExpr     ::= "always" cmpOperator numConst .
cmpOp          ::= ">" | ">=" | "<" | "<=" | "==" .
units          ::= "s" | "ms" | "us" | "ns" | "W" | "mW" | "uW" | "nW" | "%" .

reservedTokens ::= "%" | "(" | ")" | ":" | ";" | "<" | "=" | ">" | "always"
                 | "deviation" | "in" | "latency" | "likelihood" | "loss"
                 | "ms"  |  "ns"  |  "occurs"  |  "provide"  |  "readings"
                 | "require"  |  "s"  |  "sensor"  |  "specification"
                 | "throughput"  |  "tolerance"  |  "us"  |  "{"  |  "}"  .
```

Figure 5. EBNF [27] grammar for Slax, a domain-specific language for specifying latency, throughput, and value deviation tolerances for sensor access.

```
specification AccelerometerSensor;

sensor PLATFORM_ACCELEROMETER_A @ 1.6V = {
  provide (latency) {
     > 1 ms : likelihood < 1 in 1E6 readings;
  }
  provide (deviation) {
    > 1%  : likelihood < 1 in 1E6 readings;
    > 10% : likelihood < 1 in 1E9 readings;
  }
  provide (loss) {
    occurs: likelihood < 1 in 1E6 readings;
  }
}
```

```
specification PedometerApp;

tolerance PEDOMETER_TOLERANCE_ACCEL = {
  require (deviation) {
     > 1% : likelihood < 1 in 1000 readings;
  }
  require (latency) {
     > 1ms : likelihood < 1 in 1000 readings;
  }
  require (loss) {
     occurs : likelihood < 1 in 1000 readings;
  }
}
```

Figure 6. Example Slax specifications. The **`sensor`** and **`tolerance`** blocks capture sensor provisions and application requirements.

lookup table. On the other hand, efficiently picking the operating point that satisfies the multiple constraints of deviation, latency, loss, and throughput tolerances, along with timing performance, average power, and overall energy usage, will be challenging. Other challenges include:

- **Obtaining Slax `tolerance` specifications**. These could be written by hand, as in Figure 6, when there are known sensor data fidelity requirements, or could be synthesized based on dataflow analyses of applications to determine their error-propagation properties [2, 14].
- **Obtaining Slax `sensor` specifications**. As a first step, these could be constructed from manufacturer-provided data (such as in Figure 2), or from offline hardware measurements (Section 3). It would however be more versatile to be able to construct **`sensor`** specifications *in situ*, but such a facility would require appropriate hardware support.
- **Validity checking** of Slax specifications.
- **Dynamic adaptation** of the chosen operating point based on instantaneous environment conditions (e.g., temperature), based on OS or application feedback, or based on temporal histories of these.

Although we are implementing Slax using traditional compiler techniques, we are also investigating the potential benefits of integration with existing tools that ease DSL construction in systems software contexts, such as FoF [6], HAIL [21], and Termite-2 [18].

Table 1. Sensor devices evaluated, their power dissipation, and supply voltage ranges for reliable operation.

Sensor	Power Dissipation (μW)	Supply Range (V)
Gyroscope		
L3G4200D [19]	18300	2.4–3.6
IR Temperature		
TMP006B [24]	528	2.5–5.5

3. Hardware Prototype and Evaluation

To verify that the energy savings we've argued for Lax are achievable, we performed data integrity measurements at different degrees of power savings for two sensors. The sensors, listed in Table 1, are both targeted at mobile and wearable computing systems and each dissipate more power when active than the processor shown in Figure 1. In a typical system, they will also be sampled whenever the processor wakes from sleep, making their portion of the system's overall energy usage also significant.

We operated each at a range of voltages below their nominal operating points and characterized the types of errors encountered. The possible errors under these conditions are of two types: ❶ *sample loss*, or *erasures* (in the information-theoretic [5] sense), where communication with a sensor fails; ❷ *value deviations*, where values are retrieved from a sensor, but they are different from those that would have been retrieved when operating the sensor at its nominal operating voltage. Where appropriate, we modified the low-level interface code for accessing the sensors to recover gracefully from access failures (e.g., replacing assertions with more

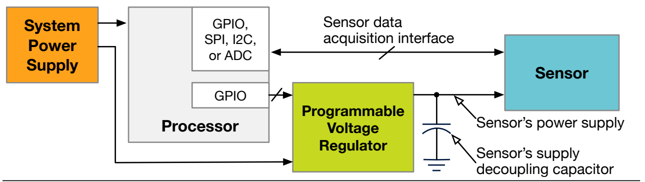

System Power Supply
GPIO, SPI, I2C, or ADC
GPIO
Processor
Sensor data acquisition interface
Programmable Voltage Regulator
Sensor
Sensor's power supply
Sensor's supply decoupling capacitor

Figure 7. Measurement setup for empirical validation of the feasibility of Lax.

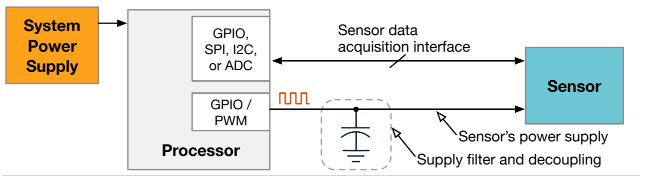


Figure 8. Programmable voltage regulators typically only output a discrete set of voltages. In some cases, they can be mimicked using a software-controlled pulse train and a filter.

graceful return status codes); this worked well for our bare-metal embedded implementation. When Lax is integrated into a sophisticated operating system, such changes might still suffice, or may be augmented with, e.g., techniques such as microreboots [4] or tools such as Carburizer [13].

3.1 Measurement setup

Figure 7 illustrates the measurement setup. We use an ARM Cortex-M0+ processor [8] evaluation board to interface with the sensors, which are mounted on separate breakout boards. The processor also controls a pair of programmable voltage regulators [25, 26] which enable the sensors to be operated at nine discrete voltages between 1.2 V and 2.5 V, with the operating voltage dynamically switchable under software control. These regulators have small circuit board footprint and low overheads, with quiescent currents in the nano-Amperes. Alternatively, the configuration shown in Figure 8 could be used to achieve even finer-grained control of sensor power supplies, with lower circuit overhead and possibly better efficiency in powering the sensor device than the programmable voltage regulator, but at the cost of additional software on the control processor.

Although our preliminary measurements use off-the-shelf evaluation boards for all the components, we have also designed (but not yet manufactured) a custom hardware platform integrating all the hardware components necessary to support Lax. We did this to validate that the necessary hardware support can be implemented in minimal printed circuit board area (less than 15 mm^2).

3.2 Preliminary results and relation to Slax

Figure 9 shows the reduction in dynamic power dissipation from operating sensors below their nominal voltages, along with the measured sensor data acquisition error rates (i.e., erasures). The measurements were performed at sensor I2C [17] interface data rates of 1 kb/s. The savings in dynamic power dissipation for both sensors are significant: up to 48% for the IR temperature sensor, and up to 42% for the gyroscope. In both cases, savings of up to 16% are possible with no data acquisition errors, and error rates increase with increasing savings. The error rate at the configuration of maximum power reduction is less than 5 errors per 100 accesses for the IR temperature sensor, but as high as 1 out of every 2 accesses for the gyroscope at maximum savings.

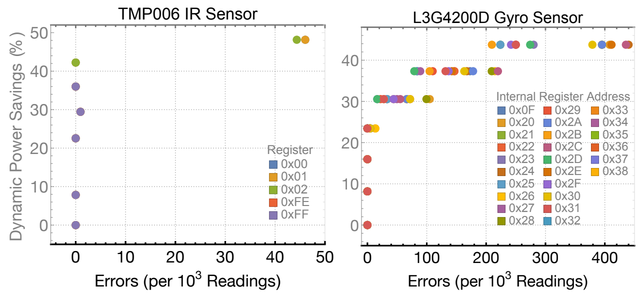


Figure 9. Power savings per access versus acquisition error rate for a state-of-the-art infrared sensor [24] (left) and gyroscope [19] (right).

The data in the figure are precisely the information required to construct a **`provide(loss){...}`** block of a **`sensor`** statement in Slax, as occurs, e.g., in the example Slax specification in Figure 6. Similarly, analysis of sensor values relative to a ground truth would enable synthesis of a **`provide(deviation){...}`** block, while data acquisition at different data rates will yield the necessary information for **`provide(latency){...}`** and **`provide(throughput){...}`** specifications.

Most sensors with digital interfaces can be queried for multiple distinct pieces of information, such as the sensor's configuration, silicon junction temperature (for software compensation algorithms), and so on. Each type of information is typically accessed via a different register internal to the sensor, and the various types of information are not equally important. The error rates observed in Figure 9 vary with which of the sensor's registers are accessed. Slax **`sensor`** specifications built from such characterization data can capture this variation through the use of separate **`sensor`** blocks for each register, or by incorporating the observed variation in the **`likelihood`** expressions.

4. Summary and Future Prospects

The energy efficiency of many embedded sensor applications can be improved by driving the electrical interfaces of sensors in a manner that makes them use significantly less power, but at the cost of unreliable data acquisition. Lax lets systems programmers exploit this insight in a controlled manner. Could even more aggressive energy-versus-fidelity tradeoffs be had with sensors? We think so, and are investigating techniques for encoding the data transferred from sensors to tradeoff the power dissipation of data transfers for reductions in sensor data fidelity.

References

[1] Analog Devices. *ADXL362 Micropower, 3-Axis,* ±2 g / ±4 g / ±8 g *Digital Output MEMS Accelerometer*, Data Sheet, 2014.

[2] F. Benz, A. Hildebrandt, and S. Hack. A dynamic program analysis to find floating-point accuracy problems. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 453–462. ACM, 2012.

[3] Bosch Sensortec. *BMX055 Small, Versatile 9-axis Sensor Module*, Data Sheet, November 2014.

[4] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot — a technique for cheap recovery. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, pages 31–44. USENIX Association, 2004.

[5] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[6] P.-E. Dagand, A. Baumann, and T. Roscoe. Filet-o-fish: Practical and dependable domain-specific languages for os development. In *Proceedings of the Fifth Workshop on Programming Languages and Operating Systems*, PLOS '09, pages 5:1–5:5. ACM, 2009.

[7] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 301–312. ACM, 2012.

[8] Freescale Semiconductor. *Kinetis KL03 32 KB Flash 48 MHz Cortex-M0+ Based Microcontroller*, Data Sheet, August 2014.

[9] Freescale Semiconductor. *MMA8451Q 3-Axis, 14-bit/8-bit Digital Accelerometer*, Data Sheet, November 2014.

[10] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 158–168. ACM, 2006.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX, pages 93–104. ACM, 2000.

[12] Janak H. Patel. CMOS Process Variations: A Critical Operation Point Hypothesis, April 2008. URL web.stanford.edu/class/ee380/Abstracts/080402-jhpatel.pdf. Stanford University Department of Electrical Engineering Computer Systems Colloquium.

[13] A. Kadav, M. J. Renzelmann, and M. M. Swift. Tolerating hardware device failures in software. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 59–72. ACM, 2009. ISBN 978-1-60558-752-3.

[14] M. D. Linderman, M. Ho, D. L. Dill, T. H. Meng, and G. P. Nolan. Towards program optimization through automated analysis of numerical precision. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '10, pages 230–237. ACM, 2010.

[15] K. Lorincz, B.-R. Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource aware programming in the pixie os. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08, pages 211–224. ACM, 2008. ISBN 978-1-59593-990-6.

[16] F. Mérillon, L. Réveillère, C. Consel, R. Marlet, and G. Muller. Devil: An idl for hardware programming. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 2–2. USENIX Association, 2000.

[17] NXP Semiconductors. UM10204, *I2C-bus specification and user manual*, April 2014.

[18] L. Ryzhyk, A. Walker, J. Keys, A. Legg, A. Raghunath, M. Stumm, and M. Vij. User-guided device driver synthesis. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 661–676. USENIX Association, 2014.

[19] ST Microelectronics. *L3G4200D MEMS Motion Sensor: Ultra-stable Three-axis Digital Output Gyroscope*, Data Sheet, December 2010.

[20] ST Microelectronics. *LPS25H MEMS Pressure Sensor: 260–1260 hPa Absolute Digital Output Barometer*, Data Sheet, January 2014.

[21] J. Sun, W. Yuan, M. Kallahalla, and N. Islam. Hail: A language for easy and correct device access. In *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, pages 1–9. ACM, 2005.

[22] Texas Instruments. *CC256x Bluetooth® and Dual-Mode Controller*, Data Sheet, January 2014.

[23] Texas Instruments. *HDC1000 Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor*, Data Sheet, November 2014.

[24] Texas Instruments. *TMP006/B Infrared Thermopile Sensor in Chip-Scale Package*, Data Sheet, November 2014.

[25] Texas Instruments. *TPS8267x 600-mA, High-Efficiency MicroSIP™ Step-Down Converter*, Data Sheet, October 2014.

[26] Texas Instruments. *TPS82740x 360nA IQ MicroSIP™ Step Down Converter Module for Low Power Applications*, Data Sheet, June 2014.

[27] N. Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM*, 20(11): 822–823, Nov. 1977.

[28] P. Zhang, D. Ganesan, and B. Lu. QuarkOS: Pushing the Operating Limits of Micro-powered Sensors. In *Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 7–. USENIX Association, 2013.

[29] N. Zhao. Full-Featured Pedometer Design Realized with 3-Axis Digital Accelerometer. *Analog Dialogue*, 44(06), June 2010.