ImpBench: A novel benchmark suite for biomedical, microelectronic implants

Christos Strydis,   Christoforos Kachris,   Georgi N. Gaydadjiev
Computer Engineering Lab, Delft University of Technology,
P.O. Box 5031, 2600 GA Delft,The Netherlands
Phone:+31-(0)15-27-83591   E-mail:christos@ce.et.tudelft.nl

Abstract*— So far, design and deployment of microelectronic, implantable devices has largely had a strongly "ad-hoc" character. The majority of existing devices has been custom-tailored to the specific application in mind, in an effort to abide by strict design constraints on safety as well as power and size. However, an enabling technology and the fact that implants are gradually becoming mainstream market products calls for a more structured design approach. Towards that end, in this paper we present ImpBench, a novel benchmark suite meant for designing and evaluating new digital processors for microelectronic implants. In an application field as wide as the various pathoses of the human body, we have conceptualized this suite based on common-sense and market-driven indicators, and we have established its usefulness and uniqueness based on extensive experimental measurement. The suite consists of eight carefully selected programs, chosen on the basis of popularity among contemporary and emerging implant applications. MiBench being the closest to our application field, that is embedded systems, has been used for a detailed comparative study. Since implants are required to perform control-, processing- or I/O-intensive tasks, various benchmark characteristics have been studied, namely: performance (IPC), cache and branch-prediction behavior, instruction distribution and power consumption. Results display significant variation from existing benchmarks to justify the need for and usefulness of ImpBench.

***Index Terms*— implant, benchmark suite, profiling, kernel, power, energy**

I. Introduction

Microelectronics design has shifted in recent years to synthesizing low-power systems. A major vehicle towards this trend has been the radical shift, through enabling technology, to portable devices such as mobile phones and laptop computers. A field of science that has adhered to strict low-power and many additional constraints since its infancy is biomedical microelectronic implants and has been around for more than 50 years. Perhaps the most popular instance of such devices is the implantable pacemaker which, apart from saving lives, has acted as a catalyst on the general public closed-mindedness against biomedical implants. Indicative of the penetration and impact pacemakers have achieved is the fact that, in Europe alone, a total number of 299,705 implanted devices have been registered over the year 2003 (source: European Society of Cardiology [1]).

With the pacemaker being the flagship, biomedical implants are now being designed for a large, and constantly increasing, range of applications. These applications are primarily grouped into two main categories: physiological-parameter monitoring (for diagnostic purposes) and stimulation (actuation, in general) [2]. Instances of the former are devices measuring body temperature [3], blood pressure [4], blood-glucose concentration [5], gastric pressure [6], tissue bio-impedance [7] and more. In the latter category belong pacemakers [8], [9] and implantable intracardiac defibrillators (ICDs) [10], various functional electrical stimulators for paralyzed extremities [11], for bladder control [12], for blurred eye cornea [13] and more pathoses.

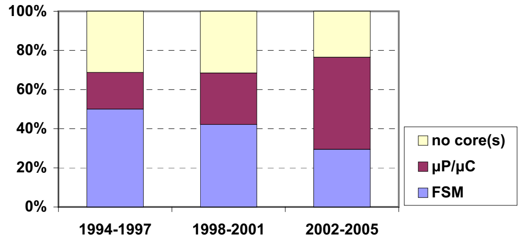


Fig. 1. Relative distribution of implant-core architecture types over the last 12 years (Source: [2]).

In a world where clinical health-care costs are increasing and population is aging, implant applications are expected to multiply in the years to come. A future where people are moving around performing their everyday tasks while tiny implants are monitoring or assisting their body in various ways is not so distant. With a market finally mature enough to embrace implants and the technological innovations of late to support them, implant designers are slowly changing their approach. With the exception of already established product cases such as the family of pacemakers introduced by Medtronic [14] where previous design expertise is (re)used to enhance the next device version, it has come to our attention that implant design has been largely custom-based; that is, implants have been developed as ASIC circuits tightly fitting the application requirements at hand. However, this is nowadays changing with implants moving from custom-designed, application-specific (e.g. FSM-based) systems [15], [16], [17] to more generic and software-based ($\mu P/\mu C$-based) ones [18], [19], [20]. This trend has been well-studied [2] and is depicted in Fig.1. What the figure tells us is that implant-processor design is becoming more streamlined and structured than it used to be and that in the near future implant functionality will be based on executed software (written in some high-level, established language like C) rather than pure, hardwired circuitry.

With the list of potential implant applications constantly expanding and the number of software-based implant solutions increasing, the need for a formal, standardized way of designing and evaluating future implant architectures becomes apparent. In this context, we have developed the ImpBench suite to address that need. While in the areas of general-purpose computing, multimedia and networking, to name a few, research has relied on well-established workload-characterization suites such as the SPEC benchmark suite [21] for optimizing the underlying hardware, this has not been the case in the area of implant-core design. Through ImpBench we target the following goals:

- In an application field which is diverse by nature, to identify a common subset of programs representative of the workloads of existing and emerging implantable systems;
- To propose self-contained programs written in a popular, high-level language so as to allow for easy porting to new implant cores under evaluation;
- To propose a benchmark suite that is freely available to the research community;
- To verify the uniqueness and, thus, usefulness of ImpBench as compared to other existing benchmark suites.

The rest of the paper is organized as follows: section II gives an overview of previously proposed benchmark suites. Section III outlines the framework onto which this study has been built, based on the characteristics of biomedical implantable devices. In section VI the details of the components of ImpBench are laid out. Section V provides the details of our selected profiling testbed for evaluating the various benchmarks. Section VI presents, reflecting upon various metrics, the contrast between ImpBench and MiBench [22], the most closely related benchmark suite to our application field. Overall conclusions and future work are discussed in section VII.

II. Related work

A large number of benchmark suites has already been proposed for various application areas. The SPEC benchmark suite with its latest version, the CPU2006 [21], targets general-purpose computers by providing programs and data divided into separate integer and floating-point categories. In particular, the design of server- and desktop-class microprocessors has been heavily influenced by the popular SPEC benchmarks as a measure of performance.

MediaBench [23], now in version II, is oriented towards multimedia- and communications-oriented embedded systems. The authors identify that most advances in compiler technology for instruction-level parallelism (ILP) have focused on general-purpose computing, driven by SPEC-characterized workloads. With the introduction and establishment of a plethora of multimedia-targeted embedded processors provisioned for increased ILP, new workloads needed to be introduced, as well. MediaBench has been put together to address that need.

The Embedded Microprocessor Benchmark Consortium (EEMBC) [24] is a non-profit organization aiming at the development of embedded-systems benchmarks for hardware and software performance evaluation. The consortium licenses ”algorithms” and ”applications” organized into benchmark suites targeting telecommunications, networking, digital entertainment, Java, automotive/industrial, consumer and office equipment products. It has also provided a suite capable of energy monitoring in the processor. Of late, EEMBC has introduced a collection of benchmarks targeting multicore processors (MultiBench v1.0). However, subject to the consortium licensing regulations, only EEMBC members are entitled to publish their benchmark test results and they can do so by previously submitting these to a certification lab.

MiBench [22] is another proposed collection of benchmarks aimed at the embedded-processor market. It features six distinct categories of benchmarks ranging from automotive and industrial control to consumer devices and telecommunications. According to the authors, MiBench has many similarities with the EEMBC benchmark suite, however it is composed of freely available source code. The diversity and usefulness of MiBench has been evaluated against the SPEC2000 benchmarks.

NetBench [25] has been introduced as a benchmark suite for network processors. It contains programs representing all levels of packet processing; from micro-level (close to the link layer), to IP-level (close to the routing layer) and to application-level programs. The authors show that although they aim architectures similar to ones MediaBench does, their workloads have significantly different characteristics. Hence, a separate benchmark suite for network processors has been considered a necessity.

Network processors are also targeted by CommBench [26], focused on the telecommunications aspect. It contains eight, computationally intensive kernels, four oriented towards packet-header processing and four towards data-stream processing. The suite is evaluated against SPEC95 and its usefulness is shown in a usage case of designing a single-chip, network multiprocessor.

III. Characteristics of implantable processors

Of late, workload-characterization programs more suited to the embedded domain have been proposed, clearly differentiating from the over-abused SPEC programs. Although in the widest sense biomedical implants are embedded systems, they adhere to a unique set of design and operation requirements, delineating their own design space and workloads. Some of the most prominent requirements are as follows.

A large class of biomedical implants performs periodic, in-vivo measurements of physiological data (blood pressure, blood temperature, intracranial pressure, blood-glucose concentration, muscle or nerve activity etc.) through appropriate sensors. The collected data need either to be stored inside the implant for later telemetry to an external monitoring device e.g. a treating physician’s office computer or to be periodically transmitted to an external data-logging system such as a PDA, laptop computer etc.. This pattern of behavior indicates that outbound biological-data traffic almost always

Compression	Encryption	Data integrity	Real applications
miniLZO [27]	MISTY1 [28]	chechsum [29]	motion [3]
Finish [30]	RC6 [28]	CRC32 [31]	DMU [19]

TABLE I
IMPBENCH BENCHMARKS.

dominates inbound traffic. Last, data must be transmitted securely as well as reliably; information eavesdropping or loss is not tolerated.

Depending on the application, implant processors may need to perform computation-, control- or I/O-intensive tasks in the human body, for instance, collection of sensory readouts, processing, storage and open- or closed-loop control of bio-actuators. In all cases, throughput should be no higher than that required by the underlying application for maintaining a low as possible energy profile and a highly reliable operation. Autonomy and dependability are primary concerns in implantable systems given the economical and health penalties involved.

Biological or other data manipulation in implants can in most cases be coped with through integer arithmetic. Expensive, floating-point operations can be avoided by smart manipulation of the data or postponed until the time when data are telemetered to an external logging station with infinite (in our context) computational resources, thus saving the implant the trouble of processing them. There are, however, distinct cases where in-vivo, run-time decisions have to be made depending on the results of floating-point math operations. One such case is simulated by the *dmu* benchmark, to be discussed in the next section.

Lastly, reported literature and an extensive study [2] on biomedical implants has further revealed that typical data-memory sizes inside the implants range from 1 KB to 10 KB. Program memories are equally restricted, with sizes in the order of magnitude of 10 KB.

IV. THE IMPBENCH COMPONENTS

Even though the previous section introduced a set of the most prominent implant characteristics, such devices have always been and will be, by nature, serving a wide variety of applications. This makes the task of identifying a representative workload set a tough one. Although ImpBench is expected to be a continuously evolving and updated tool, still we are confident that we have correctly identified a common subset of programs essential for all current and future implantable systems.

To draw a clear structure of our proposed benchmark programs, we have grouped them in four distinct categories of two programs each: *lossless data compression*, *symmetric-key encryption*, *data-integrity* and synthetic programs (what we call henceforth *real applications*). The benchmarks as summarized in Table I and are as follows:

i. **miniLZO**: MiniLZO is a light-weight subset of the LZO library (LZ77-variant). LZO is a data compression library suitable for data de-/compression in real-time, i.e. it favors speed over compression ratio. LZO is written in ANSI C and is designed to be portable across platforms. MiniLZO implements the LZO1X-1 compressor and both the standard and safe LZO1X decompressor.

ii. **Finish**: This is a C version of the Finish submission to the Dr. Dobbs compression contest. It is considered to be one of the fastest DOS compressors and is, in fact, a LZ77-variant, its functionality based on a 2-character memory window.

iii. **MISTY1**: MISTY1 is one of the CRYPTREC-recommended 64-bit ciphers and is the predecessor of KASUMI, the 3GPP-endorsed encryption algorithm. MISTY1 is designed for high-speed implementations on hardware as well as software platforms by using only logical operations and table lookups. MISTY1 is a royalty-free open standard documented in RFC2994 [32] and is considered secure with full 8 rounds.

iv. **RC6**: RC6 is a parameterized cipher and has a small code size. RC6 is one of the five finalists that competed in the AES challenge and has reasonable performance. Further, Slijepcevic et al. [33] selected RC6 as the algorithm of choice for WSNs. RC6-32/20/16 with 20 rounds is considered secure.

v. **checksum**: The checksum is an error-detecting code that is mainly used in network protocols (e.g. IP and TCP header checksum). The checksum is calculated by adding the bytes of the data, adding the carry bits to the least significant bytes and then getting the two's complement of the results. The main advantage of the checksum code is that it can be easily implemented using an adder. The main disadvantage is that it cannot detect some types of errors (e.g. reordering the data bytes). In the proposed benchmark, a 16-bit checksum code has been selected which is the most common type used for telecommunications protocols.

vi. **CRC32**: The Cyclic Redundancy Check (CRC) is an error-detecting code that is based on polynomial division. The main advantage of the CRC code is its simple implementation in hardware, since the polynomial division can be implemented using a shift register and XOR gates. In the proposed benchmark, the ITU-C CRC-16 code has been selected offering strong error correction without adding too much computation overhead. The ITU-C CRC-16 code is widely used in lightweight network protocols for wireless sensor networks such as the ZigBee protocol.

vii. **motion**: This is a synthetic benchmark based on the algorithm described in the work of Wouters et al. [3]. It is a motion-detection algorithm for the movement of animals. In this algorithm, the degree of activity is actually monitored rather than the exact value of the amplitude of the activity signal. That is, the percentage of samples above a set threshold value in a given monitoring window. In effect, this motion-detection algorithm is a smart, efficient, data-reduction algorithm.

viii. **DMU**: This is a synthetic benchmark based on the system described in the work of Cross et al. [19]. It simulates a drug-delivery & monitoring unit (DMU). This program does (and can) not simulate all real-time time aspects of the actual (interrupt-driven) system, such as sensor/actuator-specific control, low-level functionality, transceiver operation and so on. Nonetheless, the emphasis here is on the operations performed by the implant core in response to external and internal events (i.e. interrupts). A realistic model has been built imitating the real system as closely as possible.

Selection of the specific two pairs of compression and encryption algorithms, above, has been based on related works [34], [35] investigating suitable algorithms for highly resource-constrained embedded systems. As explained in those works, lossless as opposed to lossy compression algorithms have been included since information deterioration is not an option for implant applications. Also, symmetric- as opposed to asymmetric-encryption algorithms have been included since they characterize better the operational profile of implants. The checksum error-detecting code has been selected for its minimal overhead and effectiveness (it has been used in implantable systems time and again) while CRC32 has already been implemented in various light-weight network protocols including the energy-scavenging ZigBee. Lastly, we have implemented both real applications (*motion* and *dmu*) after extensively investigating the diverse field of implant applications and consider them capable of capturing commonly met operations in contemporary and future implants. Suitable datasets representing biological content have been used to feed all benchmarks. Particularly for the *dmu* benchmark, actual, field datasets have been used in order to capture the exact behavior of the simulated implantable system.

By including pairs of different algorithms performing similar functionality in ImpBench, we attempt to offer some benchmarking diversity able to capture different aspects of a new system when evaluated against the suite. This diversity will be further illustrated in section VI.

V. Experimental setup

For evaluating the uniqueness and usefulness of ImpBench, we have chosen to compare it against a number of benchmark programs extracted from MiBench. MiBench, rather than SPEC, MediaBench or other benchmarks suites, appears to be the most closely related (in terms of workloads) to the application field we are targeting. In order to perform fair comparisons between the two suits across various metrics, we had to run both benchmark collections in a suitable profiling platform.

The profiling has been based on XTREM [36], a modified version of SimpleScalar [37]. The XTREM simulator is a cycle-accurate, microarchitectural, power- and performance-functional simulator for the Intel XScale core. It models the effective switching node capacitance of various functional units inside the core, following a similar modeling methodology to the one found in Wattch [38]. XTREM has been selected for its straight-forward functionality but mostly for its high precision in modeling the performance and power of the Intel XScale core [39]. More precisely, it exhibits an average performance error of only 6.5% and an even smaller average power error of only 4% [36].

feature	value
ISA	32-bit ARMv5TE-compatible
Pipeline depth	7/8-stage, super-pipelined
Datapath width	32-bit
RF size	16 registers
Issue policy / Instr.window	in-order / single-instruction
I-Cache, L1	32KB, d.mapped (1cc-hit/170cc-miss lat.)
D-Cache, L1	32KB, d.mapped (1cc-hit/170cc-miss lat.)
TLB	1-entry fully-assoc.
BTB	2-entry direct-mapped
Branch Predictor	2-bit Bimodal (32-entry ret. addr. stack)
Write Buffer / Fill Buffer	2-entry / 2-entry
Mem. port no / bus width	1 port / 1 Byte
INT/FP ALUs	1/1
Clock frequency	2 MHz
Implem. tech.	0.18 μm @ 1.5 Volt

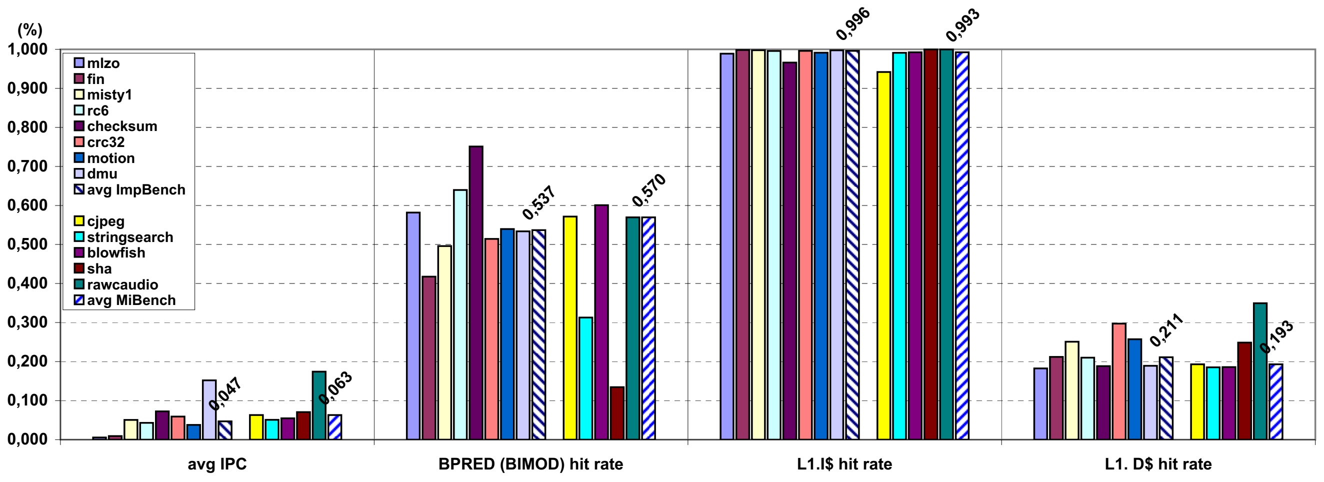
TABLE II
XTREM (MODIFIED) ARCHITECTURE DETAILS.

Many of the XScale architectural features have been integrated into XTREM. Thumb instructions and special memory-page attributes are not supported but they do not affect simulation results since they are not used by our benchmarked applications. XTREM allows monitoring of 14 different functional units of the Intel XScale core: Instruction Decoder (DEC), Branch-Target Buffer (BTB), Fill Buffer (FB), Write Buffer (WB), Pend Buffer (PB), Register File (REG), Instruction Cache (I$), Data Cache (D$), Arithmetic-Logic Unit (ALU), Shift Unit (SHF), Multiplier Accumulator (MAC), Internal Memory Bus (MEM), Memory Manager (MM) and Clock (CLK). However, to better match our application field, many of XTREM's architectural parameters have been sized down or disabled to better reflect the highly constrained implantable processors. The modified XTREM characteristics are summarized in Table II. Performance/power figures have been checked and scale properly with the changes.

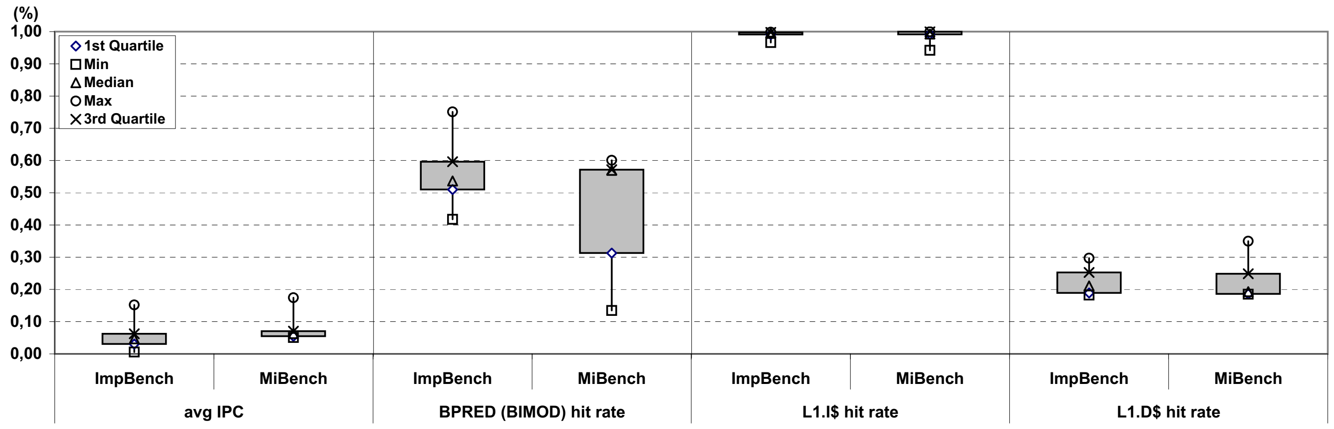
VI. Benchmark characterization

Since the MiBench suite spans a wide range of application fields, we have selected a small but representative subset from the ones most related to our own field. Selection has also been based on porting issues since not all MiBench programs could be successfully compiled on our bare-metal (i.e. no OS-support) simulator. From the "Consumer" category *jpeg* has been been chosen, from the "Office" category *stringsearch*, from the "Network" category *blowfish*, from the "Security" category *SHA* and from the "Telecomm." category *ADPCM enc.*. For all profiled benchmarks, only the compression part of compression algorithms and the encryption part of cryptographic algorithms have been considered since they are the most computationally demanding aspects and/or are the most commonly executed from the point of view of implantable systems.

The goal of this phase is to empirically test whether the ImpBench suite is quantitatively different from the MiBench suite, each operating on its own representative datasets (inter-benchmark variation). We also wish to point out the variety in behavior offered by the two alternative flavors in each one of the four ImpBench categories (intra-benchmark variation). Most results indicate relative values, that is, ratios. Unless otherwise stated, discussed average values are, in fact,



(a) Per-benchmark, average values.



(b) Box-and-whiskers plot.

Fig. 2. IPCs, I-/D-cache hit rates and branch-prediction rates

median values which are more suitable since collected data are not guaranteed to be normally distributed in the general case.

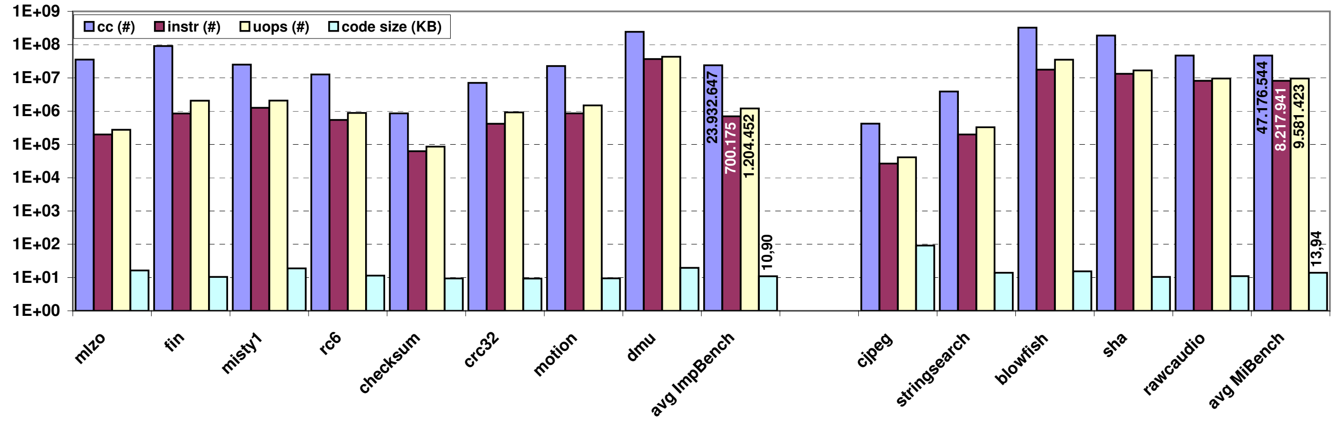
A. *Performance, caches and branch prediction*

The first characteristic we explore is benchmark performance measured as the Instructions Per Cycle (IPC) of each benchmark. Since IPC depends also on the cache performance and the efficiency of the branch-prediction unit, we include such results here as well. Accordingly, in Fig. 2, overall average IPC, L1 I-cache and D-cache hit rates and branch-prediction rates are depicted.

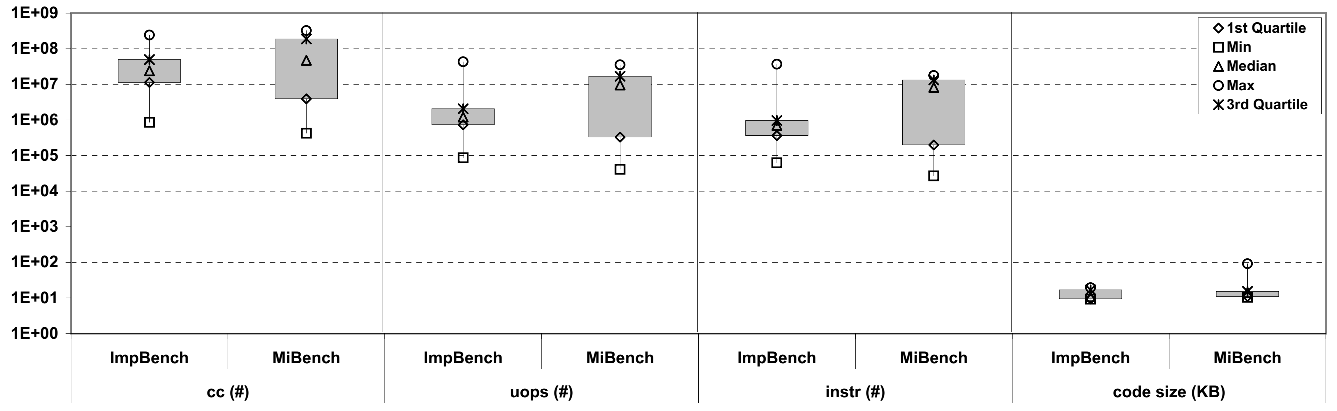
As can be seen from Fig. 2(a), ImpBench programs achieve on average a lower IPC (0.047) than the MiBench ones (0.063); yet, both IPCs are expectedly low. To elaborate, in order to closely model real implantable processors, the XTREM simulator has been modified to such a degree that all tasks running on it are effectively ”choked” by the limited resources left on it. That is, the intrinsic performance of many tasks is capped by the maximum performance the simulated processor can deliver. This is reflected in the limited IPCs observed for both benchmark suites. However, Fig. 2(b) captures a more prominent difference between the two suits. Although both suite distributions are skewed closer to their minimum values, ImpBench programs display a wider dispersion as can be seen by the box sizes formed by the 1st and 3rd quartile (i.e. the middle 50% of the values).

In terms of intra-benchmark variation, MiBench's *rawcaudio* (ADPCM encoding) and *sha* perform apparently better than most of the ImpBench programs while *stringsearch* is scoring the lowest in MiBench and even low for many ImpBench programs. The two compression algorithms of ImpBench, although varied, display by far the poorest performance across all benchmarks, seemingly impacted by the limited D-cache size, as will be discussed later. For the encryption algorithms, *misty1* appears to perform better than *rc6* while, for the data-integrity algorithms, *checksum*'s simpler structure clearly outperforms *crc32*. Last, *motion*, of the real applications, although simpler, performs significantly worse than *dmu* contrary to which *motion* displays a higher ratio of I/O- over control- or data-intensive operations. In effect, it reads successive data from a file (representing motion-sensor readouts), compares them against a preset motion threshold value and writes an activity factor to an output file (representing a data-logging memory).

As shown in the previous section, a Bimodal branch-prediction scheme has been used with a mere 2-entry table, the reasons being: a) to reflect the constrained nature of an implant processor and, b) to isolate the dynamic behavior



(a) Per-benchmark, average values.



(b) Box-and-whiskers plot.

Fig. 3. Static code size (in KB) and dynamic code size (instruction and clock-cycle count).

of the various benchmarks. Referring back to Fig. 2, we conclude that overall branch-prediction (BPRED) hit rates are similar for both suites. However, contrary to the observed IPCs, the MiBench programs display a significantly wider dispersion of BPRED values than ImpBench. Further, the MiBench values are strongly skewed towards the maximum value whereas ImpBench values present a distribution closer to the Gaussian. In effect, ImpBench programs present a slightly less predictable but overall more consistent dynamic behavior among them and it is interesting to note that the range (i.e. max-min) of ImpBench BPRED values (0.333) is quite smaller than that of MiBench ones (0.466).

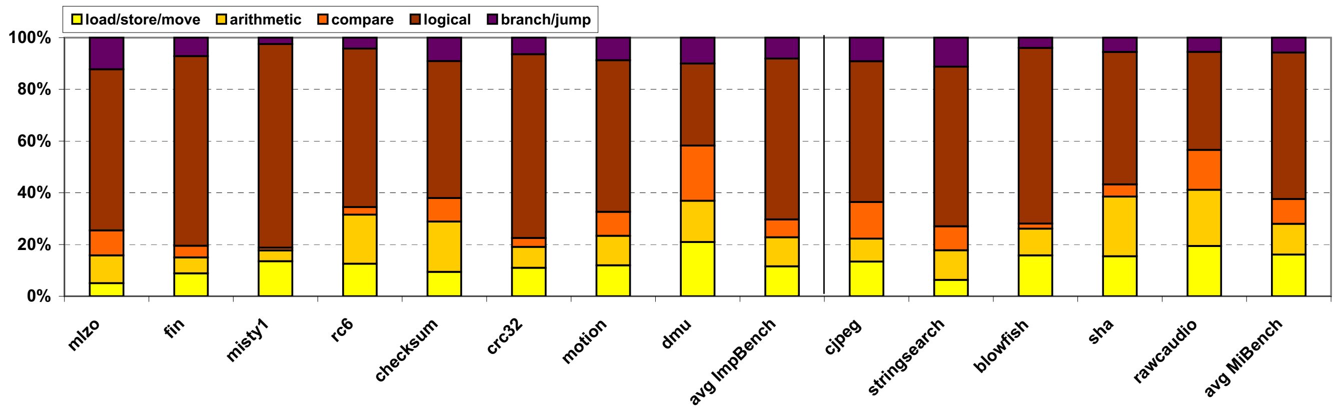
Besides, *fin* achieves a worse BPRED rate than *mlzo* and the worst overall for ImpBench programs but, still, an almost x3 better rate than the worst MiBench program (*sha*). In terms of intra-variation, encryption and data-integrity algorithms also vary largely in behavior while the real applications display similar profiles.

The selected I-cache configuration and the intrinsic behavior of the programs has yielded essentially miss-free cache operation, as can be seen in Fig. 2(a). This behavior is observed in the MiBench programs as well, featuring a marginally smaller I-cache hit rate. Figure 2(b) indicates that, in this case, value dispersion is extremely low with a slight skew towards the maximum value, for both suites. Combined, the two figures tell us that in terms of I-cache behavior, there is no significant difference between the two suites.

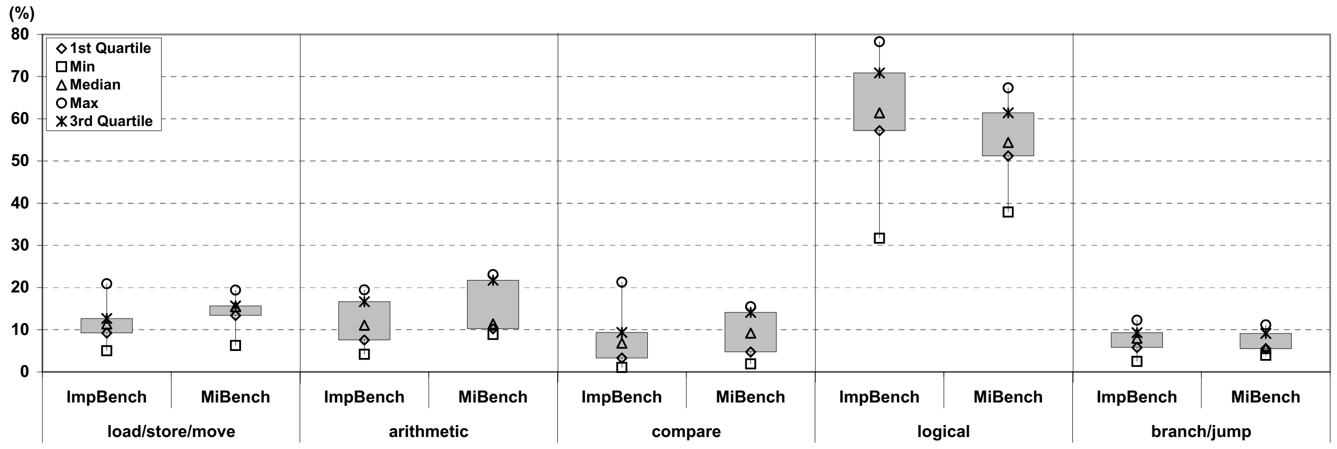
The D-cache, on the other hand presents a different hit-rate behavior. As seen in Fig. 2(a), ImpBench programs feature an overall 0.211 miss rate, quite higher than its 0.193 MiBench counterpart, but both much lower than the I-cache hit rates witnessed previously. Figure 2(b) further reveals that the dispersion of values is moderate for both suites, with ImpBench being marginally larger. Yet, its distribution is again closer to the Gaussian than that of MiBench whose values are clearly skewed towards the minimum. A last observation to make at this point is that the lower hit rates of the D-cache, as compared to the I-cache, reveal a strong data-intensive nature of the biomedical applications.

B. Dynamic & static benchmark size

We, next, delve into the differences between the two benchmark suites in terms of static and dynamic program size. From Fig. 3 we can readily observe that, in an overall, ImpBench programs feature a moderately smaller static size, about $10.9\ KB$ compared to the $13.94\ KB$ of the MiBench programs but their sizes are somewhat more dispersed. *cjpeg* displays the overall largest static size while *crc32* the overall smallest one. In terms of intra-variation, *fin* is smaller than *mlzo*, *rc6* is smaller than *misty1* while *checksum* is slightly larger than *crc32*. *dmu* is much larger than *motion* which is to be expected since the former implements a much larger and more complex application. We can also notice that across the



(a) Per-benchmark, average values.



(b) Box-and-whiskers plot.

Fig. 4. Relative frequencies for load/store/move, arithmetic (int/fp), compare, logic and branch/jump instructions.

```
repeat for all consecutive instruction triplets of the program {
    let instr1, instr2, instr3 be 3 new consecutive instructions.
    if (instr2.src_reg1 == instr1.dest_reg) or (instr2.src_reg2 == instr1.dest_reg)
        instr2 is dependent on instr1 (pair)
    if (instr3.src_reg1 == instr1 dest_reg) or (instr3.src_reg2 == instr1.dest_reg)
        also instr3 is dependent on instr1 (triplet)
} end
```

TABLE III
INSTRUCTION-DEPENDENCY ALGORITHM.

compression and encryption categories, the algorithms which perform better, do so at an increased code size.

In terms of dynamic behavior, which depends on the input datasets used as well as on the intrinsic structure of the various benchmarks, we can observe that ImpBench programs exhibit, on average, a clock-cycle count about half that of the MiBench ones but a much smaller dispersion of values. This, once more, reveals the more diverse nature of the biomedical applications selected. We have also plotted the total number of executed instructions and μops. XTREM, which is based on SimpleScalar, implements ARM instructions through μops. We included μop statistics at this point and in the following discussion so as to better capture the workings of the underlying architecture. Overall, ImpBench programs display shorter execution times (about $0.5x$) but much shorter instruction/μop counts (about $0.1x$) than the MiBench ones. This agrees with the observations we made previously regarding the IPC and attests to the more control- and I/O-intensive nature of the biomedical programs compared to general multimedia programs. Last, it is interesting to observe that, although *fin* and *crc32* have smaller static code sizes than their respective counterparts *mlzo* and *checksum*, they have much larger dynamic code sizes. Given that all compression, encryption and data-integrity algorithms operate on the same input dataset (a 10-KB binary file of ECG readouts), these observed variations between static and dynamic program sizes directly expose diverse intrinsic properties of the various algorithms.

C. Instruction distribution

Having discussed overall instruction counts, we elaborate further on the nature of the executed instructions per benchmark suite, i.e. the instruction mix. For the same reason as before, we choose to profile μops rather than instructions. We

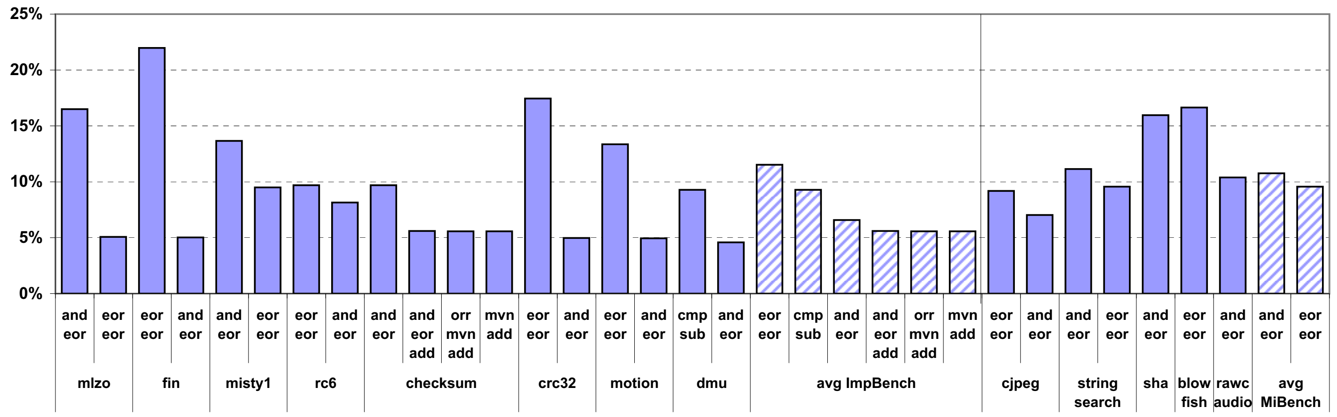


Fig. 5. Relative frequencies of data-dependent, dynamic-instruction combinations.

have organized μops into five groups: data move (load, store, move), arithmetic operations (INT and FP), comparison operations, logical operations and branches (conditional and unconditional). Only the *dmu* benchmark includes floating-point operations and even that does not stress them. We wish to adhere to the initial observation that the majority of implant applications can do without or with few floating-point calculations. In effect, FP arithmetic operations are scarce or absent and have, thus, been merged with the INT arithmetic operations. Overall μop mixes are shown in Fig. 4. We can readily observe that rates of ld/st/mov, arith, cmp and logical operations all differ significantly between the ImpBench and MiBench programs. Overall, we notice less ld/st/mov and cmp μops for ImpBench. Yet, there are more logical and br/j μops further supporting the argument that biomedical applications exhibit a more dynamic behavior than average multimedia ones. The number of arith μops is similar for both suites. We also observe that, all μop categories display a larger range (max-min) of values for ImpBench compared to MiBench. In terms of data dispersion, the ImpBench boxplots reveal more dispersed ld/st/mov and logical μop ratios.

In terms of intra-variation, *mlzo* differs largely from *fin* in all μop ratios, *misty1* differs notably from *rc6* in arith and logical μop ratios and, with the addition of cmp ratios, so does *crc32* compared with *checksum*. All in all, the variation among the various ImpBench programs is visible. A last observation to make at this point is that logical μops are clearly dominating the μop mix which is partly explained by the known tendency of the utilized ARM cross-gcc to favor the generation of logical instructions.

The XTREM simulator has been further modified to also collect pairs and triplets of data-dependent instructions during execution time. Data-dependent instructions have been defined according to the simple algorithm shown in Table III. In effect, with the previous algorithm we are scanning for all data-dependent dynamic instructions of the program. What we are interested in is the exact nature of those pairs or triplets of dependent instructions. In Fig.5 we have plotted dependent-instruction combinations for all profiled benchmarks. We have limited the plot to only those combinations appearing with a frequency of 4.5% or higher during dynamic-code execution. With this limitation, Fig. 5 has been plotted. It reveals that ImpBench and MiBench both favor (heavily depending on the compiler used) predominantly the ”and-eor” and ”eor-eor” pairs with high occurrence frequencies (”eor”: exclusive-or operation). ImpBench once more illustrates a higher diversity and introduces more frequent instruction combinations. Namely, the pairs ”cmp-sub” (due to *dmu*) and ”mvn-add” (due to *checksum*) as well as the triplets ”and-eor-add” and ”orr-mvn-add” (both due to *checksum*). As desired, it also captures different instruction dependencies within the compression, the encryption, the data-integrity and the real-programs categories.

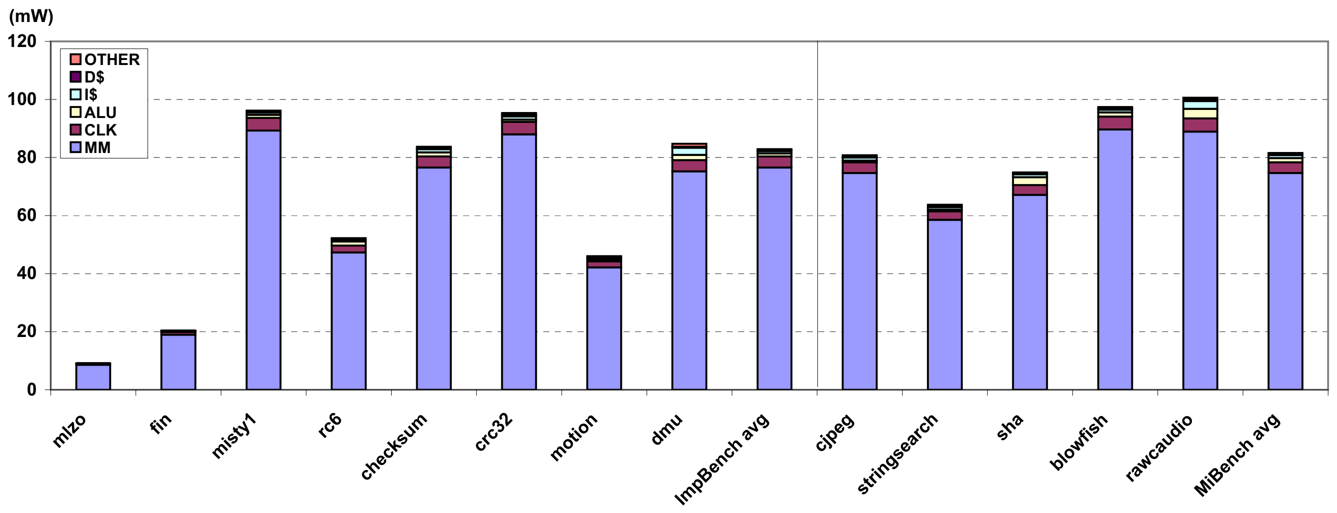
This ”instruction-combination” metric has been monitored since it can prove beneficial for the design of future implant processors. When popular instruction combinations (along with the previously discussed single-instruction frequencies) have been identified, specific microarchitectural or architectural optimizations can be made to achieve more efficient machines. For instance, in a processor design seriously limited by power and area constraints, to favor the execution of ”eor-eor” or ”and-eor” pairs, only data forwarding in the logical-operations circuitry of the ALU may be allowed to be incorporated. Alternatively, other techniques like the known interlock-collapsing ALUs [40] can also be considered.

D. Power consumption

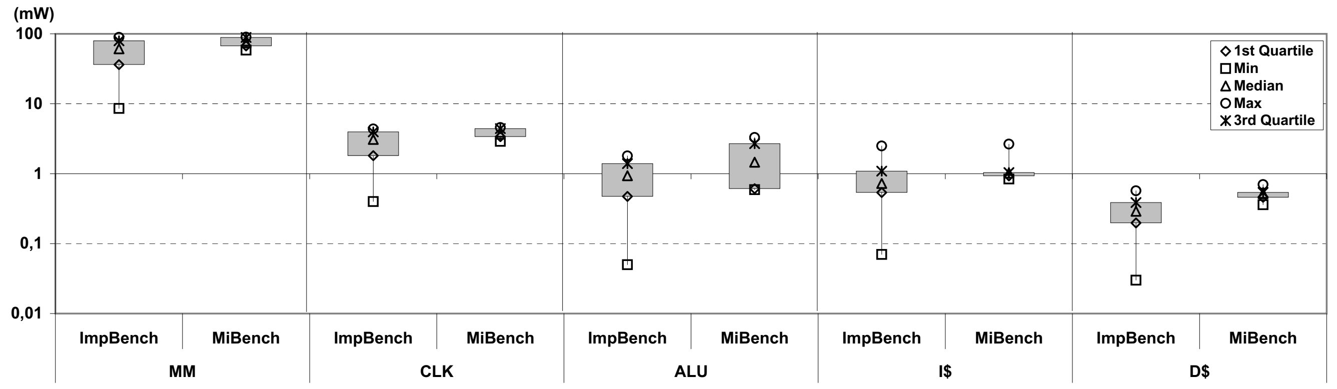
A final metric to be evaluated against ImpBench and MiBench is average power consumption of the executed programs. This is highly relevant to embedded systems and particularly to energy-scavenging microelectronic implants.

In Fig. 6 we have plotted the per-component and overall average power consumption of our simulated processor. Overall, the average power consumption of MiBench is about 1.2x times that of ImpBench. This can be attributed partly to the fact that most ImpBench programs have been carefully picked for low-power applications [35], [34]. Yet, it also indicates that they can provide meaningful means of workload characterization for implant processors since they implicitly respect tight power budgets ever present in the considered application field.

Further analysis of the results indicates the most power-hungry unit to be the memory manager (MM) for ImpBench and MiBench programs alike, followed by the CLK, ALU, I-cache and D-cache structures. MiBench manages to stress



(a) Per-benchmark, average values.



(b) Box-and-whiskers plot.

Fig. 6. Per-component and overall average power consumption.

the power profile of the MM more than ImpBench. The same is true for the ALU and I-cache. However, the ImpBench programs display, in all components except for the ALU, more data-dispersed power profiles. This finding further enforces the initial observation that biomedical programs indeed are more diverse in characteristics than general multimedia ones.

In terms of intra-benchmark variation, we can clearly see that the compression algorithms display by far the smallest power consumption across both suites. Last, with the exception of the two data-integrity algorithms, the two members of all other ImpBench categories vary largely between them in terms of power, i.e. one features an average power consumption half or double that of the other.

VII. Conclusions

A plethora of benchmark suites has been proposed so far for a variety of application domains. Of late, workload-characterization programs more suited to the embedded domain have began spurting in an attempt to better capture the particular characteristics of embedded processors.

A special category of embedded systems with particular design constraints is implantable, microelectronic devices. Since their birth, such devices have been traditionally designed in custom style, always attempting to squeeze the desired functionality in an extremely limited size and with the maximum possible safety. With the advent of mature microelectronics and micromachining technologies as well as more sophisticated computer-architecture and compiler design, this trend has began to change. Continuously more implant designers are willing (and free) to move to software-running implant architectures to achieve their goals.

In view of more structured and educated implant processors in the years to come, we have carefully put together ImpBench, a collection of benchmark programs and assorted input datasets, able to capture the intrinsic of new architectures under evaluation. We have shown that ImpBench displays considerably different characteristics than the most related MiBench. IPCs, data-cache hit rates, branch-prediction hit rates, instruction frequencies and power consumption show increased or sufficient variation compared to MiBench, to justify a new benchmark suite.

ImpBench is a dynamic construct and, in the future, more benchmarks will be added, subject to our ongoing research. Among others, we anticipate simple DSP applications as potential candidates as well as more ”real applications” like the ones we already included.

VIII. Acknowledgements

This work has been partially supported by the ICT Delft Research Centre (DRC-ICT) of the Delft University of Technology. It would not have been complete without the valuable contributions of Gilberto Contreras for providing an excellent power simulation tool and support as well as of Peter Cross for providing the original sources for the DMU application. Also, many thanks are due to Carlo Galuzzi for his valuable comments and help throughout.

References

[1] H. Ector and P. Vardas, "Current use of pacemakers, implantable cardioverter defibrillators, and resynchronization devices: data from the registry of the european heart rhythm association," *European Heart Journal Supplements*, vol. 9, pp. 144–149, August 1988.

[2] C. Strydis, G. Gaydadjiev, and S. Vassiliadis, "Implantable microelectronic devices: A comprehensive review," Computer Engineering, Delft University of Technology," CE-TR-2006-01, December 2006.

[3] P. Wouters, M. D. Cooman, D. Lapadatu, and R. Puers, "A low power multi-sensor interface for injectable microprocessor-based animal monitoring system," in *Sensors and Actuators A: Physical*, vol. 41-42, 1994, pp. 198–206.

[4] B. Flick and R. Orglmeister, "A portable microsystem-based telemetric pressure and temperature measurement unit," in *IEEE Transactions on Biomedical Engineering*, vol. 47, Jan. 2000, pp. 12–16.

[5] M. Shults, R. Rhodes, S. Updike, B. Gilligan, and W. Reining, "A telemetry-instrumentation system for monitoring multiple subcutaneously implanted glucose sensors," in *IEEE Transactions on Biomedical Engineering*, vol. 41, Oct. 1994, pp. 937–942.

[6] P. Valdastri, A. Menciassi, A. Arena, C. Caccamo, and P. Dario, "An implantable telemetry platform system for in vivo monitoring of physiological parameters," in *IEEE Transactions on Information Technology in Biomedicine*, vol. 8, Sept. 2004, pp. 271–278.

[7] M. Min, T. Parve, V. Kukk, and A. Kuhlberg, "An implantable analyzer of bio-impedance dynamics - mixed signal approach," in *IEEE Instrumentation and Measurement*, Budapest, Hungary, 21-23 May 2001, pp. 38–43.

[8] J. Berkman and J. Prak, "Biomedical microprocessor with analog I/O," in *IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, 19 February 1981, pp. 168–169.

[9] C. Harrigal and R. Walters, "The development of a microprocessor controlled implantable device," in *IEEE Proceedings of the 1990 Sixteenth Annual Northeast Bioengineering Conference*, Mar. 1990, pp. 137–138.

[10] J. Warren, R. Dreher, R. Jaworski, J. Putzke, and R. Russie, "Implantable cardioverter defibrillators," in *Proceedings of the IEEE*, vol. 84, 1996, pp. 468–479.

[11] B. Smith, Z. Tang, M. Johnson, S. Pourmehdi, M. Gazdik, J. Buckett, and P. Peckham, "An externally powered, multichannel, implantable stimulator-telemeter for control of paralyzed muscle," in *IEEE Transactions on Biomedical Engineering*, vol. 45, 1998, pp. 463–475.

[12] M. Sawan, S. Robin, B. Provost, Y. Eid, and K. Arabi, "A wireless implantable electrical stimulator based on two FPGAs," in *Proceedings of the IEEE International Conference on Electronic Circuits and Systems (ICECS)*, vol. 2, Piscataway, New Jersey, USA, 1996, pp. 1092–1095.

[13] M. Schwarz, L. Ewe, R. Hauschild, B. Hosticka, J. Huppertz, S. Kolnsberg, W. Mokwa, and H. Trieu, "Single chip CMOS imagers and flexible microelectronic stimulators for a retina implant system," in *Sensors and Actuators A: Physical*, vol. 83, 22 May 2000, pp. 40–46.

[14] "Medtronic - Cardiology product list," http://www.medtronic.com/physician/cardiology.html.

[15] M. Ghovanloo and K. Najafi, "A modular 32-site wireless neural stimulation microsystem," in *IEEE Journal of Solid-State Circuits*, vol. 39, December 2004, pp. 2457–2466.

[16] P. Mohseni and K. Najafi, "Wireless multichannel biopotential recording using an integrated FM telemetry circuit," in *26th Annual International Conference of the IEEE in Engineering in Medicine and Biology Society (EMBS)*, San Francisco, CA, USA, 1-5 September 2004, pp. 4083–4086.

[17] H. Park, H. Nam, B. Song, and J. Cho, "Design of miniaturized telemetry module for bi-directional wireless endoscopy," *IEICE Transactions on Fundamentals on Electronics, Communications and Computer Sciences*, vol. 6, pp. 1487–1491, June 2003.

[18] C. Liang, J. Chen, C. Chung, C. Cheng, and C. Wang, "An implantable bi-directional wireless transmission system for transcutaneous biological signal recording," *Physiological Measurement*, vol. 26, pp. 83–97, February 2005.

[19] P. Cross, R. Kunnemeyer, C. Bunt, D. Carnegie, and M. Rathbone, "Control, communication and monitoring of intravaginal drug delivery in dairy cows," in *International Journal of Pharmaceuticals*, vol. 282, 10 September 2004, pp. 35–44.

[20] H. Lanmuller, E. Unger, M. Reichel, Z. Ashley, W. Mayr, and A. Tschakert, "Implantable stimulator for the conditioning of denervated muscles in rabbit," in *8th Vienna International Workshop on Functional Electrical Stimulation*, Vienna, Austria, 10-13 September 2004.

[21] "SPEC CPU2006," http://www.spec.org/cpu2006/.

[22] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *IEEE International Workshop on Workload Characterization*, pp. 3–14, 2 December 2001.

[23] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," *30th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 330–335, 1-3 Dec 1997.

[24] "EEMBC," http://www.eembc.com.

[25] G. Memik, W. H. Mangione-Smith, and W. Hu, "NetBench: a benchmarking suite for network processors," in *IEEE/ACM international conference on Computer-aided design (ICCAD'01)*, Piscataway, NJ, USA, 2001, pp. 39–42.

[26] T. Wolf and M. Franklin, "CommBench-a telecommunications benchmark for network processors," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'00)*, Washington, DC, USA, 2000, pp. 154–162.

[27] M. Oberhumer, "LZO v2.0.2," http://www.oberhumer.com/opensource/lzo/.

[28] Y. Law, J. Dourmen, and P. Hartel, "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, pp. 65–93, February 2006.

[29] R. Braden, D. Borman, and C. Partridge, "Computing the internet checksum," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 2, pp. 86–94, 1989.

[30] N. de Vries, "Lossless Data-Compression Kit, LDS v1.3," http://www.nicodevries.com/nico/lds13.zip.

[31] "Cell Relay Retreat: CRC-32 Calculation, Test Cases and HEC Tutorial," http://cell.onecall.net/cell-relay/publications/software/.

[32] H. Ohta and M. Matsui, *A Description of the MISTY1 Encryption Algorithm*, United States, 2000.

[33] S. Slijepcevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, and M. Srivastava, "On communication security in wireless ad-hoc sensor networks," *Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'02)*, pp. 139–144, 2002.

[34] C. Strydis and G. Gaydadjiev, "Profiling of lossless data-compression algorithms for a novel biomedical-implant architecture," in *To appear in IEEE/ACM International Conference on Hardware-Software Codesign and System Synthesis (CODES'08)*, Atlanda, Georgia, USA, 19-24 October 2008.

[35] C. Strydis, D. Zhu, and G. Gaydadjiev, "Profiling of symmetric encryption algorithms for a novel biomedical-implant architecture," in *ACM International Conference on Computing Frontiers (CF'08)*, Ischia, Italy, 5-7 May 2008, pp. 231–240.

[36] G. Contreras, M. Martonosi, J. Peng, R. Ju, and G.-Y. Lueh, "XTREM: A Power Simulator for the Intel XScale Core," in *LCTES'04*, 2004, pp. 115–125.

[37] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, February 2002.

[38] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *ISCA'00*, 2000, pp. 83–94.

[39] *Intel XScale Microarchitecture for the PXA255 Processor: User's Manual*, Intel Corporation, March 2003.

[40] S. Vassiliadis, J. Phillips, and B. Blaner, "Interlock collapsing ALU's," *IEEE Transactions on Computers*, vol. 42, no. 7, pp. 825–839, Jul 1993.