J Sign Process Syst (2009) 57:107–119
DOI 10.1007/s11265-008-0255-x

Design of 100 μW Wireless Sensor Nodes for Biomedical Monitoring

Lennart Yseboodt · Michael De Nil · Jos Huisken · Mladen Berekovic · Qin Zhao · Frank Bouwens · Jos Hulzink · Jef Van Meerbergen

Received: 3 April 2008 / Accepted: 4 July 2008 / Published online: 17 October 2008

Abstract Wireless sensor nodes span a wide range of applications. This paper focuses on the biomedical area, more specifically on healthcare monitoring applications. Power dissipation is the dominant design constraint in this domain. This paper shows the different steps to develop a digital signal processing architecture for a single channel electrocardiogram application, which is used as an application example. The target power consumption is 100 μW as that is the power energy scavengers can deliver. We follow a bottleneck-driven approach: first the algorithm is tuned to the target processor, then coarse grained clock-gating is applied, next the static as well as the dynamic dissipation of the digital processor is reduced by tuning the core to the target domain. The impact of each step is quantified. A solution of 11 μW is possible for both radio and DSP running the electrocardiogram algorithm.

Keywords Wireless sensor node · ASIP · Low power · Clock gating · ECG

L. Yseboodt (✉) · M. De Nil · J. Van Meerbergen
Eindhoven University of Technology, Den Dolech 2,
5612 AZ Eindhoven, The Netherlands
e-mail: lennart.yseboodt@philips.com

M. De Nil
e-mail: Michael.DeNil@imec-nl.nl

J. Huisken
Stichting IMEC Nederland, HTC 31,
5656AE Eindhoven, The Netherlands
e-mail: jos.huisken@imec-nl.nl

M. De Nil · M. Berekovic · Q. Zhao · F. Bouwens · J. Hulzink
Stichting IMEC Nederland, High Tech Campus 31,
5656 AA Eindhoven, The Netherlands

L. Yseboodt · J. Van Meerbergen
Philips Research Eindhoven, High Tech Campus 37, 5656
AA Eindhoven, The Netherlands

J. Van Meerbergen
e-mail: jef.van.meerbergen@philips.com

1 Introduction

A new generation of biomedical monitoring devices is emerging. The main challenge for this kind of devices is low power dissipation. In this context a power budget of only 100 μW is available for the whole system including radio, digital processing and memories. This power is taken from extremely small batteries with or without energy scavengers. To reduce the power dissipation of the radio data compression or feature extraction is used to reduce the number of bits that must be transmitted. Thus the bottleneck shifts towards the digital part which is the focus of this paper.

The goal of our work is to create a low-power C-programmable DSP, optimized for the application domain via hardware support for application specific instructions. As starting point a reconfigurable processor from Philips' technology incubator Silicon Hive [11] is selected. This technology includes a retargetable C compiler making code development and portability for these processors easy. This programmability is important because of the wide range of applications that can run on the nodes. Programmable nodes allow a lower non-recurring engineering cost for the software and the hardware.

We differentiate between static and dynamic power dissipation. The dynamic power is the power consumed due to switching and the internal power, which is the power used inside the cells due to short-circuit currents and all the power used in the internal nets. It includes the functional units, memories, controller and clock. Current CMOS technology trends indicate that leakage is becoming more dominant with every new process generation. In our experiments leakage power soon turns out to be an important factor, up to 100 μW of leakage was measured. Our focus has gone both in reducing static as dynamic power by minimizing the time the processor is active. As a case study we examined an ECG algorithm running on the proposed platform, what we learned from this example led to more general system level conclusions.

2 System Level Architecture

A generic sensor node consists of several subsystems as depicted in Fig. 1. There is a digital processing subsystem with level 1 local memory, a level 2 memory subsystem, including RAM and non-volatile memories, an array of sensors and possibly actuators, a radio system and a power subsystem including a source and powermanager, which is responsible for waking up various parts of the node when needed. This conceptual model holds independent of specific chip or die boundaries and leaves open several packaging technologies. If level 2 memories are kept off-die then multiple instances of the sensor node can be made without having to create a new chip.

In current systems the power is supplied by a small battery or from energy scavengers. Battery powered nodes have the disadvantage of requiring maintenance. Different forms of energy scavenging are possible but in this paper we assume a power budget of around 100 μW [5]. This number includes power consumed by the radio and the sensors, it is the global power budget of the entire sensor node.

The digital subsystem must be programmable in order to be able to run different algorithms such as ECG or EEG analysis, or altogether new algorithms from the

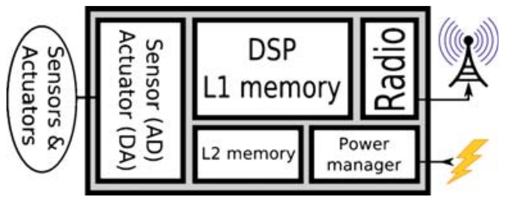


Figure 1 Overview of the architecture of a wireless sensor node.

biomedical domain. Furthermore real time constraints must be met especially when actuators are involved.

From a power dissipation point of view the most important consumers are the radio, the memory and the digital subsystem. Commercially available radios consume 150 nJ/bit [13] and as a consequence the transmission of raw data can be expensive. An algorithm to reduce the amount of data via compression or feature extraction usually is a better compromise between computation and communication. In addition to the radio most subsystems exploit duty cycling and sleep modes to reduce the dissipation. Next the DSP must be tuned to the application. Also the memory subsystem can dissipate a lot of power. What is needed is a hierarchical memory subsystem optimized for power dissipation by reducing the size of the lowest level memories. These design principles will now be discussed in more detail and illustrated with an example, which is explained first.

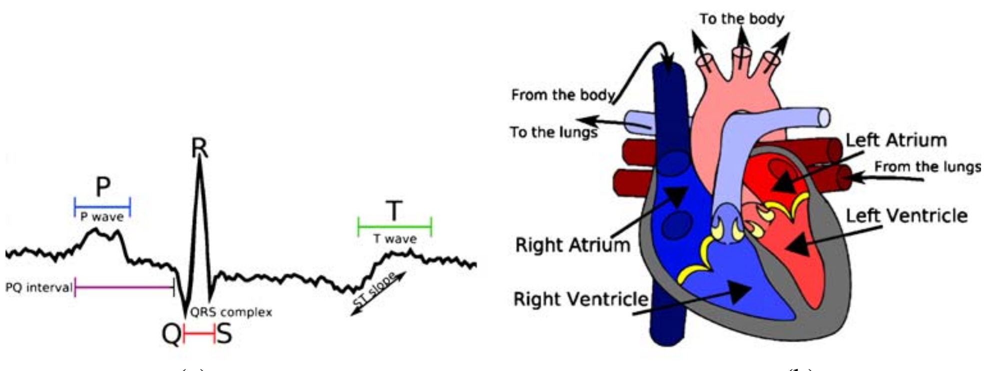
3 Electrocardiogram

The electrocardiogram is a well studied topic, for which several interesting algorithms exist. An excerpt of a real ECG signal is shown in Fig. 2a with the most important points and events shown. The rhythm of the heart is controlled by the sinoatrial node. Starting at the resting point oxygen-rich blood from the lungs and oxygen-deprived blood from the body is flowing into the two atria. The P wave indicates the opening of the valves between the atria and the ventricles. The PQ interval gives the blood time to flow into the ventricles. At point Q the ventricles contract. This causes the QRS complex, the well know ‘peak’ of the electrocardiogram. After this contraction the muscles depolarize. This is visible as the small T wave. The blood flow is depicted in Fig. 2b.

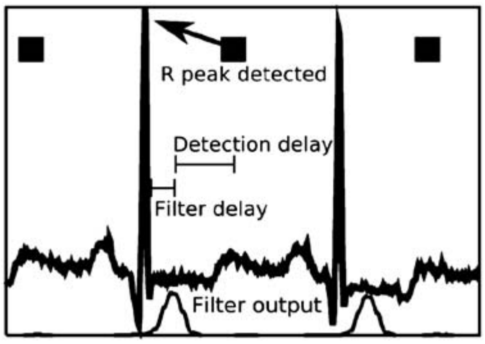
One of the most basic functions of electrocardiogram analysis is the detection of the R peak. The location of this peak is relevant because it allows to calculate the heart rate. Furthermore if this location is known it is less complicated to search for the other elements that are harder to detect.

The algorithm we use is based on the opensource ECG detection program from EP Limited [2–4]. This code uses the Pan-Tomkins [10] method for R peak detection. The Pan-Tomkins method is a filtering based method to detect the frequency that is unique to the steeper R peak.

The design of the code is straightforward. The raw ECG samples from the sensor are first passed through a series of filters. The intent is to get rid of all details except the frequency of the QRS complex. The results



(a) (b)



(c)

Figure 2 The electro-cardiogram and the blood flow (**a**) an excerpt of a real ECG signal (**b**) blood flow (**c**) results of the filter stage and beat detection.

of the filter stage is seen as the bottom line in Fig. 2c. All other elements have been removed and only the QRS complex is still visible after this step. Next, this result is presented to a beat detection algorithm, which is a simple state machine that decides whether an R peak is detected based on a variable offset trigger and the time the previous R peak was detected.

There are 5 steps in the filtering process. The data is channeled through a low-pass filter, a high-pass filter, a derivative, absolute value and integrator function, in that order. The effects of each filter stage are depicted in Fig. 3. The low-pass filter removes the small artifacts. It eliminates high frequency noise, even in the case of a powerful magnitude. To demonstrate this a strong high frequency noise was added to the inputfile (the black rectangle). After the low-pass filter stage it is still visible but very small. The high-pass filter removes most of the slow artifacts caused by breathing or movement. After this stage the ECG is mostly flat except for the QRS complex itself. From what remains the derivative is taken. Since the rising and falling edges of the QRS complex are very steep, it leads to a high and a low peak in the derivative. The other waves will be attenuated further. The ABS is needed for the integration stage, without this stage the positive and negative peak would cancel each other out. The integration stage calculates the average of the waveform over a small period of time, thereby smoothing out the peaks. After this stage the QRS complex is visible as a smooth 'hill' all other features and artefact's are attenuated strongly.

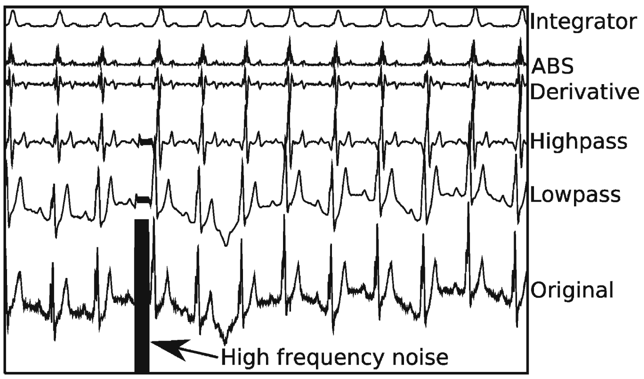


Figure 3 Output of the various filter stages.

$$lp(x)_y = 2y[n-1] - y[n-2] + x[n] - 2x[n-5] + x[n-10] \tag{1}$$

$$hp(x)_z = x[n-13] - y[n] \tag{2}$$

$$y[n] = y[n-1] + x[n] - x[n-26] \tag{3}$$

$$der(x) = x[n] - x[n-2] \tag{4}$$

$$int(x) = \sum_{a=0}^{a<16} x[a]/16 \tag{5}$$

$$filter(x) = int(abs(der(hp(lp(x))))) \tag{6}$$

These filter equations are only valid for a sample frequency of 200 Hz.

A certain delay is caused by the filters and there is further delay caused by the decision making stage in order to avoid false positives. A false positive could occur when there are strong T waves. This delay varies between 300 ms and 500 ms. These two delays can be see in Fig. 2c. The state machine code is only run when a beat is detected. On average the filter code is running 99% of the time.

For reasons explained later, it is beneficial to run the algorithm in batches. Because of the way the filters were designed it was most efficient to have the filter code run on a batch of 50 samples. This also made the filtering code slightly faster because the compiler had more scheduling freedom.

Extra software was written to provide additional parameters. From the location of the R peak the location of Q and S can be calculated with a simple algorithm. This allows the calculation of the duration of the QRS complex, an important parameter in ECG analysis. Furthermore statistics on R to R intervals are kept and also an average heartbeat over a certain amount of beats. This extra information comes at an additional cycle cost. To have a consistent benchmark, only the basic analysis code has been used in the calculations.

4 Implementation Using State-of-the-Art DSP

In order to do local processing, a digital signal processor is required on the wireless sensor node. In this section four modern processors, deemed suitable for use in wireless sensor nodes, are benchmarked with the ECG application described in the previous section.

4.1 Atmel ATMega

The Atmel ATMega128L [1, 9] is part of the Atmel AVR family, a set of 8 bit RISC microprocessors. The ATMega128L has a flash program memory of 128 kB, which is in-system reprogrammable, 4 kB EEPROM and 4 kB internal SRAM. The processor can operate between 2.7 V and 5.5 V, and can be clocked with a frequency up to 8 MHz. In our benchmark the processor was powered with 2.7 V and clocked at 8 MHz. When compiling the ECG code described in Section 3 using AVR-GCC, the average amount of execution cycles required to process one sample was 1223 cycles. This means that the microprocessor will have an activity of 224.600 cycles per second. Power measurements show a total power consumption of 1.363 mW. This was achieved by putting the ATMega128L in the low power "Standby Mode" when no processing was required. Measurements were done on an STK500 + STK501 development board. The power consumption of the board was determined by doing power measurements without a processor in the socket. This power number was subtracted from the total power measured with a processor.

4.2 TI MSP430

The TI MSP430F149 [12] is designed as a 16 bit Von Neumann architecture with combined instruction and data bus, and is known for it's very low idle power. The MSP430F149 has a non-volatile program memory of 60 kB, which is in-system programmable, and 2 kB internal SRAM. The processor can operate between 1.8 V and 3.6 V, and can be clocked with a frequency up to 4.15 MHz at 1.8 V and 8 MHz at 3.6 V. In our benchmark the processor was powered with 1.8 V and clocked at 32.768 kHz. When compiling the ECG code with msp430-gcc, the average cycle count per sample is 988 cycles, meaning that the processor is active for 197.600 cycles per second. Power measurements show a total power consumption of 204 μW. This was achieved by putting the MSP430F149 in "Low Power Mode 4" during idle. The power measurements were performed on an custom PCB with only a 32.768 kHz and the MSP430F149 on board (Table 1).

Table 1 Overview platforms.

	ATMega128L	MSP430F149	Coolflux	PearlRay
Voltage	2.7 V	2.7 V	1.2 V	1.2 V
Frequency	8 MHz	32.768 kHz	Core: 48 MHz IO: 12 MHz	100 MHz
Data bus size	8 bit	16 bit	24 bit	32 bit
Program memory size	128 kB	60 kB	256 kB	32 kB
Data memory size	4 kB	2 kB	DM X: 144 kB DM Y: 48 kB	32 kB

Table 2 Comparison of state-of-the art signal processors running ECG algorithm.

	Active power	Idle power	ECG IPS	Total power
Atmel ATMega128L (8 MHz - 2.7 V)	20.6 mW	785 μW	244600	1.363 mW
TI MSP430F149 (32.768 kHz - 2.7 V)	31.9 μW	∼ 0 μW	197600	204 μW
NXP Coolflux (48 MHz - 1.2 V)	1.3 mW	2.23 mW	68800	2233 μW
Silicon Hive PearlRay (100 MHz - 1.2 V)	66.97 mW	858 μW	51900	861 μW

4.3 NXP Coolflux

The NXP CoolFlux [8] is designed as a Dual-Harvard architecture containing two semi-independent 24-bit datapaths (X and Y), with one shared program counter. Intermediate addition and multiplication results are stored in 56 bit accumulator register files. The processor is designed for low power applications in mind, and requires a power supply voltage between 0.7 and 1.2 V, depending on the required clock frequency. In our benchmark the processor core was clocked at 48 MHz, the IO was clocked at 12 MHz and the whole system was powered with 1.2 V. The CoolFlux development board comes with connectors which allow direct measurement of all processor currents without overhead of other components on the test board. The Target Compiler Toolkit was used as a compiler to map the ECG application. With the compiled simulator Checkers, an average cycle count of 344 cycles per sample was obtained. When running the ECG application on the system, a total power consumption of 2.23 mW was measured. Due to an error in the processor supplied we were not able to switch to idle mode. This caused a very high idle power consumption compared to the other processors.

4.4 Silicon Hive PearlRay

The Silicon Hive PearlRay [6] is a 3 issue slot C programmable VLIW processor. This processor was not supplied on Silicon, but power results were obtained using worst case commercial power simulations on a netlist after route simulation. In this simulation the processor is clocked at 100 MHz, has an 128 bit wide instruction memory of 2048 words and a 32 bit wide data memory of 8192 words. The processor has fine grained clock gating, but does not consist of special idle power modes. A power simulation on the PearlRay processor running the ECG application showed a power consumption of 861 μW, taking into account that the processor had a lower power consumption while idle. The compiler supplied with Silicon Hive, HiveCC, obtained an average cycle count of 260 cycles.

5 Digital Signal Processor Optimization

5.1 Reference Core

Table 2 shows that current state-of-the art processors are not directly usable for this application, due to our power constraints. Although the Silicon Hive PearlRay has a higher idle power compared to the Atmel AVR or the TI MSP430, it is the most viable starting point to create a 100 μW solution. This since this processor has a very high power efficiency (Table 3) and is reconfigurable, i.e. there exists a parameterizable description of the architecture and a C-compiler that can generate code for any possible architecture instance supported by the processor template.

The top level configuration file controls certain aspects of the processor: data widths, functional unit placement, custom functional units, configurations of the issue slots... We generated a default configuration with 32 kB of data memory and 32 kB of program memory. The processor is a VLIW with three issue slots, 128 bit wide instructions and is synthesized for a speed of 100 MHz. This speed is the ‘sweet spot’ for this design. Synthesizing the core for several clock frequencies shows that speeds above 100 MHz make the design grow exponentially in area and leakage as depicted in Fig. 4.

The algorithm was optimized by recoding the filters in such a way that their behavior was largely unaffected, when several expensive divisions were replaced by shifts. The PearlRay does not have a hardware divider and relies on a software divider taking 25 cycles per division. After these optimizations the cost of analyzing one sample of ECG data at a 200 Hz sampling frequency was 250 cycles, however when a beat is detected

Table 3 Power efficiency of state-of-the art signal processors running ECG algorithm.

	Voltage	μW/MHz	MIPS/mW
Atmel ATMega128L	2.7V	2.575 μW/MHz	0.4
TI MSP430F149	2.7V	974 μW/MHz	1.0
NXP Coolflux	1.2V	103 μW/MHz	9.7
Silicon Hive PearlRay	1.2V	69.7 μW/MHz	14.3

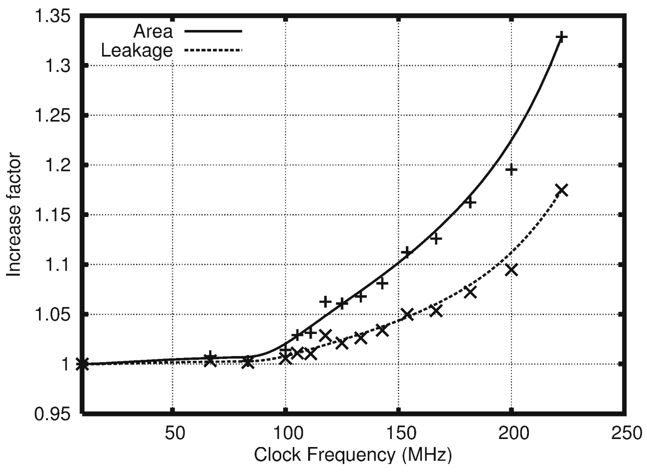


Figure 4 Clock frequency vs. Area & Leakage.

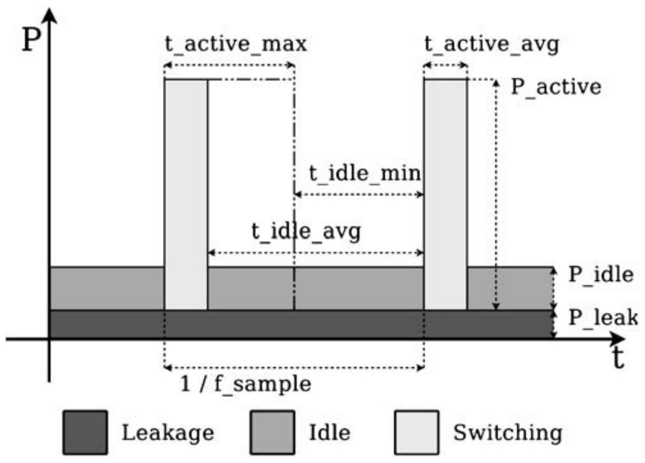


Figure 6 Causes of power consumption over the time domain.

this number is higher: 1200 cycles. A detection of a beat occurs only once or twice every second so on average it takes $198 \cdot 250 + 2 \cdot 1200 = 51900$ cycles per second. If the PearlRay is running at 100 MHz the duty cycle is $51900/100 \cdot 10^6 = 0.05\%$.

Power figures for the processors, as seen in Table 5, were obtained using Synopsys PrimePower with layout extracted capacitances. As input a vector file from a netlist simulation was used, which was generated using Cadence Ncsim. Simulations were based on the processor netlist after layout on a 90 nm CMOS process (Fig. 5).

The power dissipation of the PearlRay was analyzed first. Three modes are identified: active, idle and sleep. In active mode the processor is running a program and processing samples. In idle mode the clock is still running. In sleep mode the only dissipation is due to leakage.

Graphically sketched this is visible in Fig. 6. In this diagram power consumption is plotted versus time. The area of the bars represents the energy consumed. The lightest bars represent the active energy, which can vary dependent on the input sample. We also observed this behavior in our ECG software. The middle bar is the idle energy and the darkest block is the ever present leakage energy.

$$P_{\mathrm{Tot}} = P_{\mathrm{Leak}} + f_{\mathrm{sample}}((P_{\mathrm{Act}} \cdot t_{\mathrm{Act\_avg}}) + (P_{\mathrm{Idle}} \cdot t_{\mathrm{Idle\_avg}})) \tag{7}$$

The ECG application is an example of an algorithm that does not require a large portion of the processing power that the reference core offers. Therefore the developed processor is optimized for algorithms with a low duty cycle. Table 4 shows the power characteristics of the standard version of the PearlRay, which is used as a reference. At first glance the active power is dominant, but since the processor is only ‘active’ for a small fraction of the time, the actual energy usage attributed to active mode constitutes only to 0.4% of the total energy consumption (Table 5). The power used in idle mode is the dominant factor here.

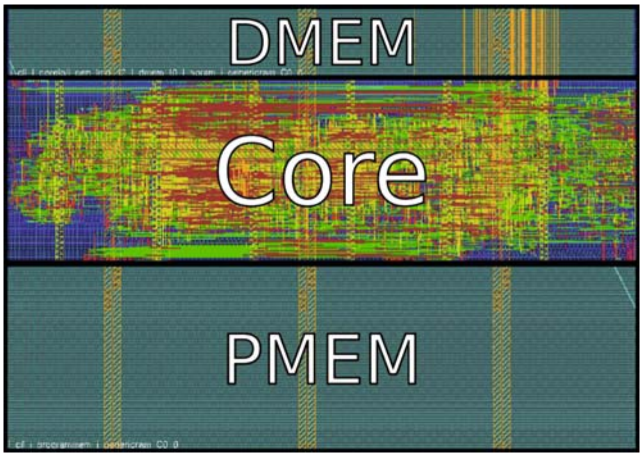


Figure 5 Layout of the optimized PearlRay core.

Table 4 Standard version of the PearlRay used as a reference.

Source	Power	Duration	Mean power
Active	6.87 mW	496 μW	3.41 μW
Idle	0.76 mW	1s−496 μs	758 μW
Leak	100 μW	1 s	100 μW

The last column shows the energy for one input sample and one ECG computation. Total power 861.4 μW

Table 5 PrimePower output results for reference PearlRay while active.

	P_Switch	P_Int	P_Leak	P_Total	%
imec_ref	1.46e-3	5.41e-3	1.00e-4	6.97e-3	100
core	9.11e-4	7.78e-4	9.53e-6	1.70e-3	24.4
dec	3.86e-5	1.75e-4	2.34e-7	2.13e-4	3.1
is_I0	1.00e-4	4.21e-5	9.67e-7	1.44e-4	2.1
is_I1	2.71e-4	1.56e-4	2.80e-6	4.30e-4	6.2
is_I2	8.96e-5	5.64e-5	9.61e-7	1.47e-4	2.1
rf_I0	4.69e-5	9.61e-5	1.35e-6	1.44e-4	2.1
rf_I1	1.03e-4	8.30e-5	2.07e-6	1.88e-4	2.7
rf_I2	3.24e-5	5.32e-5	8.07e-7	8.65e-5	1.2
coreio	2.21e-4	1.11e-3	5.01e-5	1.38e-3	19.8
genI1	2.69e-6	3.45e-5	7.15e-7	3.79e-5	0.5
genI2	3.54e-5	5.92e-5	2.69e-7	9.49e-5	1.4
genI3	1.47e-6	6.69e-5	1.38e-6	6.98e-5	1.0
pmem	4.14e-5	3.37e-3	3.90e-5	3.45e-3	49

The `coreio` contains the data memory.

5.2 Reduce Idle Mode Dissipation

To counter the effects of idle energy we use coarse grained clockgating. The PearlRay reference core was already using fine grained low-level clock gates but the top level clock gate was not implemented. The top level clock gate disconnects the clock from the entire clock-tree, meaning that when this gate is open no switching will occur in the processor. As a consequence an external piece of circuitry must revive the processor when this is required. Such a clock gate was very important as shown by the results in Table 6. After this optimization the dominant energy component is leakage (96%).

5.3 Reducing Leakage

Now we are faced with dominant leakage power so we analyze in which part of the processor the leakage occurs. Our total leakage is 100 μW, of which 50 μW is caused by the data memory, 40 μW by the program memory and 10 μW by the processor itself. The large majority of the leakage is in the memories. We tried four things to improve this leakage.

– Reduce the size of that data memory to 2 kB. Since the ECG program only requires 1.2 kB and 120 bytes of stack this was possible. This reduced the leakage to 65.6 μW, a 34.5% improvement.
– By removing one of the three issue slots in the PearlRay processor and reducing the size of the immediates, the width of the program memory could be reduced from 128b to 64b. Due to the decrease of parallelism the instruction count was increased with 27%, but the instruction width was reduced by 50%, allowing us to reduce the program memory from 32kB to 16kB. This resulted in a reduction of leakage power to 82μW, a 18% improvement.
– The use of memory modules designed in a technology with a high threshold option (HighV_t). This drastically reduces the leakage of the memories. They will become slower but speed was not really a constraint and the memories still operated on 100MHz. Using these memories leakage was reduced to 16.2μW, a 84% improvement.
– Reduce the datapath from 32 bit to 16 bit. As the samples are only 16 bit wide and all operations occur on them, it is optimal to scale the core to this width. This gave a moderate improvement in leakage to 94.7μW, or 5.3%.

When combining these techniques together with floorplan optimizations, the results shown in Table 7 were obtained, which reduced the leakage of the original PearlRay processor to 5.45μW, a 94.5% improvement. Furthermore scaling down the datapath to 16 bit also contributed to reduce the dissipation of the active mode.

Table 6 Power results with a top level clockgate installed.

Source	Power	Duration	Mean power
Active	6.87 mW	496 μs	3.41 μW
Idle	0 W	1 s–496 μs	0 W
Leak	100 μW	1 s	100 μW

Total power 103.41 μW.

Table 7 Power result with anti-leakage techniques combined.

Source	Power	Duration	Mean power
Active	4.7 mW	628 μs	2.95 μW
Idle	0 W	1 s–628 μs	0 W
Leak	5.45 μW	1 s	5.45 μW

Total power 8.4μW.

6 System Level Optimization

In this section we describe system level optimizations that are a work in progress. We are currently experimenting with power gating and level 2 memories that can be used to save the state and shutdown the core.

6.1 Power Down the System

From Table 7 we conclude that the leakage is still dominant. Therefore an interesting option is to power down the core and to save the state to level 2 memory and restore it when the next batch of samples have to be processed. There are positive and negative contributions to the power dissipation. In those circumstances where the final net result is positive this is an interesting option. It means a hierarchical memory subsystem: small level 1 memories with a high number of accesses and larger level 2 memories with a very limited number of accesses. This is similar to a memory hierarchy in computer architectures but optimized for power dissipation instead of performance. Level 2 memory (or part of it) is also used for other purposes, e.g. to collect the samples that arrive while the core is down or to store multiple applications, which are not active simultaneously.

Let's apply this to the ECG example. The state includes not only data (1.2 kB) but also the program (16 kB). This data is used to retain the state of the filters and for several other variables such as the baseline drift. An important decision is the granularity of switching between modes. If we do this at a sample basis this can become quite expensive. Assuming a low power (level 2) SRAM memory in a 90 nm process and a size of 32 kB the cost of an access is 0.875 pJ/B and the leakage equals 2.5 μW. If the processor is powered down after every sample the cost is 28.8μW, the calculation is detailed in Table 8. This can be improved by grouping the samples in groups of 50, then the cost of saving and restoring is also reduced by a factor 50 which translates into an acceptable level of 3.0 μW. This can even be further improved to 0.5 by using a non-volatile memory (flash).

The swapping between level 2 and level 1 memories can be done for complete applications but also for parts of an application. The Pan-Tomkins algorithm for ECG is a good example. As mentioned above it consists of 2 parts: the filtering and the feature extraction. Both parts have similar code size. The filtering is executed for every sample but the feature extraction is executed with a low probability (0.5%), i.e. only when a beat is detected, which is about once per second. Therefore it is possible to reduce the level 1 code memory by a factor of 2, which reduces the access energy. The consequence is that the programmer or the compiler must be aware of this, e.g. to insert statements for code swapping.

Table 8 Level 1 to level 2 state save calculation.

Cause	Calculation	Result
Granularity: 1 sample		
Leak		2.5 μW
R_{pm}[a]	16kB·8192·0.875pJ·200/s	22.94 μW
W_{st}[b]	1200B·8· 0.875pJ· 200/s	1.68 μW
R_{st}[c]	1200B·8· 0.875pJ· 200/s	1.68 μW
	Total:	28.8 μW
Granularity: 50 samples		
Leak		2.5 μW
R_{pm}	16kB·8192· 0.875pJ· 4/s	0.46 μW
W_{st}	1200B·8· 0.875pJ· 4/s	0.03 μW
R_{st}	1200B·8· 0.875pJ· 4/s	0.03 μW
	Total:	3.02 μW

[a]Read program memory
[b]Write state
[c]Read state

6.2 Powergating Tradeoff

Power gating is a technique that allows at runtime for components to be turned off. This is done by installing a PMOS transistor at the supply rail or a NMOS transistor between the subsystem and the ground. It became obvious during the optimizations to the PearlRay processor that the largest portion of the power was consumed by leakage in the memories. It would therefore be interesting to be able to shutdown these memories during the idle period. The state is then saved in external flash or SRAM, as is explained in Sections 6.3 and 6.1.

Now that we know that powergating can save energy by removing most of the leakage in the system, we would like to find out in what cases exactly it is interesting to apply this technique. Designs with power gates consume more active power than designs without power gate technology. This extra active power is caused by the dissipation of the power switches. This voltage drop can be limited by putting multiple power switches in parallel. This causes however also a lower resistance when the switch is turned off, and thus a higher leakage when the system is idle. Furthermore it should be noted that when a component is powered on, there will be energy required to load the capacitance of the system and there will be additional powerloss due to possible short-term short currents in the circuit.

The power to just leave everything on is equal to

$$P_{active} \cdot T_{active} + P_{leak} \cdot T_{leak} \qquad (8)$$

whereas the power consumed in a scenario with power gating is

$$(\mathrm{P}_{\mathrm{active\_powergate}} + \mathrm{P}_{\mathrm{leak}}) \cdot \mathrm{T}_{\mathrm{active}} + \frac{\mathrm{Bootcost}}{\mathrm{Granularity}} \qquad (9)$$

$\mathrm{T}_{\mathrm{leak}}$ is the fraction of the time that the design leaks, which is 1. $\mathrm{T}_{\mathrm{active}}$ is the fraction of the time that the processing is working, or the duty cycle. The increased active power number due to the power gates is signified by $\mathrm{P}_{\mathrm{active\_powergate}}$. In these scenarios one can reduce the effect of the Bootcost by trying to keep the devices in a shutdown state for as long as possible. There might be latency constraints on this. The factor by which this rebooting can be delayed is denoted as Granularity.

For our ECG application we have chosen a granularity of 50 samples for technical and latency related issues. The design of the filter code allowed modifications such that 50 samples (or multiples thereof) could be processed in one batch with a minimum of processing overhead. Adding the delay of 50 samples (250 ms), the delay of the filters and beat detection gave a beat detection latency of just under 1 s which is acceptable for this application.

Rewriting the formula for given power numbers and an unknown break-even granularity

$$\mathrm{Bootcost} \cdot \left(\mathrm{T}_{\mathrm{active}} \cdot (\mathrm{P}_{\mathrm{active}} - \mathrm{P}_{\mathrm{active}^p} - \mathrm{P}_{\mathrm{leak}}) + \mathrm{P}_{\mathrm{leak}} \cdot \mathrm{T}_{\mathrm{leak}}\right) \qquad (10)$$

or rewriting for a given granularity the maximum cost of starting the processor is

$$\frac{\mathrm{T}_{\mathrm{active}} \cdot (\mathrm{P}_{\mathrm{active}} - \mathrm{P}_{\mathrm{active}^p} - \mathrm{P}_{\mathrm{leak}}) + \mathrm{P}_{\mathrm{leak}} \cdot \mathrm{T}_{\mathrm{leak}}}{\mathrm{Granularity}} \qquad (11)$$

Since we do not have figures for the cost of starting a PearlRay from power down, the only extra information we can calculate is the maximum power that a bootup can consume before it is no longer interesting to powergate the processor and better to let it leak instead. As it was not possible to simulate the effects of power gating on power consumption we ignored this factor. The actual number depends on the design and the area-power trade-off in power gating technology.

When shutting down the memories or the processor the state of that subsystem has to be saved. This data transfer and retrieval also consumed a portion of the power. It is explained more in the next subsection. For this trade-off we assume that this cost is contained in the general 'bootup' cost.

Table 9 Elements that play a role in the SRAM or Flash trade-off.

Description		
SRAM leakage	$\mathrm{S}_{\mathrm{leak}}$	a
Datasize	$\mathrm{D}_{\mathrm{size}}$	b
Read/write cost SRAM	$\mathrm{Srw}_{\mathrm{cost}}$	c
Programsize	$\mathrm{P}_{\mathrm{size}}$	d
Write cost flash	$\mathrm{FW}_{\mathrm{cost}}$	e
Save&restore/sec		f
Read cost flash	$\mathrm{FR}_{\mathrm{cost}}$	h
Boot flash chip	$\mathrm{F}_{\mathrm{boot}}$	g

6.3 Flash or SRAM for State Saving

A final trade-off has to be made when deciding where to save the state. Again we will see that this is largely dictated by the application. More specifically, if the power-down period is large enough then it will be worth saving the state to flash, otherwise SRAM is preferred.

Several factors come into play, they have been given shorthand names as shown in Table 9.

The cost of saving state to SRAM is

$$\mathrm{S}_{\mathrm{leak}} + (2\mathrm{D}_{\mathrm{size}} \cdot \mathrm{Srw}_{\mathrm{cost}} + \mathrm{P}_{\mathrm{size}} \cdot \mathrm{Srw}_{\mathrm{cost}}) \cdot f \qquad (12)$$

while the cost of writing to flash is

$$(\mathrm{D}_{\mathrm{size}} \cdot \mathrm{FW}_{\mathrm{cost}} + \mathrm{D}_{\mathrm{size}} \cdot \mathrm{FR}_{\mathrm{cost}} + \mathrm{P}_{\mathrm{size}} \cdot \mathrm{FR}_{\mathrm{cost}} + \mathrm{F}_{\mathrm{boot}}) \cdot f \qquad (13)$$

Rewriting the formula to find the break even point (in time) between flash and sram yields

$$\frac{1}{f} = \frac{be + bh + dh + g - 2bc + cd}{a} \qquad (14)$$

By calculating this break even point the maximum amount of state-saves per seconds are allowed before the greater cost of reading and writing flash is more than the leakage of the SRAM chip. Section 6.1 (in Table 8) showed an example of the effect of choosing the right granularity on the effectiveness of using this technique.

6.4 Program Overlay

As can be seen from the results obtained from Table 8 for DSPs with a wide program memory the size of the program can quickly become critical. A possible improvement is to partition the program into smaller blocks which are loaded when needed. The ECG algorithm for example consists roughly of a filtering part and a decision making part. In terms of code size they are approximately equal. The filtering part is needed for every sample, but the decision making portion of the code is for the most part only needed in 1% of the cases. By only loading the part of the program that is needed in the ECG example case up to 50% of power can be saved in terms of useless copying of part of the program code at the expense of having additional code complexity. In larger, more complex programs these gains could even be greater as in most cases 90% of the functionality is provided by 10% of the code. The programmer would then be responsible to load the parts of the code that are needed.

6.5 Results

Table 10 shows a system level overview of the different components of the power consumption in μW. Furthermore the application scope is widened. The first four rows show an ECG application with different assumptions. The first row shows the simple baseline ECG case with 1 channel as discussed above. The second row assumes 3 channels. The next one is again 1 channel but now a more complex algorithm for a more extensive analysis including extra parameters (such as Q&S peaks and average beat rate). The fourth one is the same as the previous one but now for 3 channels. The last two rows show FFT analysis on 1 and 10 channel(s) respectively. The different columns represent the different contributions to the power dissipation in μW. A 90 nm process is assumed. The second column represents the radio power assuming 150 nJ/bit. Columns 3 and 4 are related to the processor and show the dissipation when active and the leakage. The next column shows the dissipation due to state-saving and restoring in a 32 KB level 2 SRAM memory. The last 2 columns show the total dissipation for 2 different scenarios. The last column assumes level 2 memory is used and the processor put in power down mode. The previous column assumes the opposite.

We conclude for various use scenarios different components can have the largest contribution in power consumption. Therefore it is not easy to predict and a careful analysis is needed for each situation. The data in Section 5 shows that the average power consumption constraint of 100μW is feasible.

Table 10 Power consumption with different assumptions, all numbers represent micro watts.

	P_{radio}	P_{active}	P_{idle}	P_{state}	$P_{tot\ L1}$	$P_{tot\ L1,L2}$
1ch	4.8	3.3	5.5	3.0	13.5	11.0
3ch	4.8	9.8	5.5	3.1	20	17.6
1ch+	9.6	4.6	5.5	3.3	19.6	17.5
3ch+	9.6	13.7	5.5	3.6	28.7	26.8
eeg1	2.16	2.1	5.5	3.0	9.8	7.3
eeg10	21.6	21.3	5.5	3.0	48.4	45.9

7 Conclusion

Power dissipation is the most important constraint for wireless sensor nodes for healthcare applications. This paper describes the different steps in the development of an architecture using a single channel ECG application as an example. It shows that a 100 μW solution is feasible.

For minimum power dissipation there is an optimum balance between computation and communication. Transmitting raw data is usually not optimal. A significant reduction in the amount of transmitted bits is obtained via compression or feature extraction. As a consequence the bottleneck shifts towards the DSP. Static as well as dynamic dissipation must be tackled. Both components are reduced by tuning the core to the target domain (application specific instructions, proper memory sizes, etc.) In an optimized architecture the level 1 memories have a limited size due to the high number of accesses in active mode. When the processor is inactive it can be powered down while the state is saved in level 2 memory. This requires that the granularity is carefully chosen. Analyzing different ECG applications it is shown that optimizing the digital processing technology is important.

Therefore this is chosen as the focus of this paper. Using ECG as a driver and adopting a bottleneck-driven step-by-step approach a factor of 100 reduction of power dissipation of the DSP core was measured via simulations. This is a result of the following actions that span the different design levels.

- Algorithm level: optimization and simplification of the code.
- Architecture level: e.g. level 1 memory size reduction by a factor of 2 for instructions and a factor of 16 for data
- Gate level: e.g. clock gating.
- Technology with HighV_t.

References

1. Atmel (2008). Atmel ATMega128L Datasheet. www.atmel.com.
2. De Nil, M., Yseboodt, L., et al. (2007). *Ultra low power ASIP design for wireless sensor nodes*. IEEE ICECS. Piscataway: IEEE.
3. Ekanayake, V. N., Kelly IV, C., & Manohar, R. (2005). BitSNAP: Dynamic significance compression for a low-energy sensor network asynchronous processor. In *Proc. ASYNC* (pp. 144–154), March.
4. EP Limited (2008). EP Limited homepage. http://www.eplimited.com.
5. Gyselinckx, B. (2008). *Human^{++}: Emerging technology for body area networks*. Boca Raton: CRC.
6. Halfhill, T. R. (2003). Silicon hive breaks out. Microprocessor report. www.MPRonline.com. Accessed 1 Dec 2003.
7. Meyr, H. (2003). System-on-chip for communications: The dawn of ASIPs and the dusk of ASICs. In *Proc. IEEE Workshop on Signal Processing Systems (SIPS'03)*. Seoul, Korea, Aug 2003.
8. NXP (2007). NXP Coolflux DSP. www.coolfluxdsp.com.
9. Pan, J., & Tompkins, W. J. (1985). A real-time QRS detection algorithm. *IEEE Transactions Biomedical Engineering, BME-32*(3), 230–236.
10. Rangayyan, R. (2002). *Biomedical signal analysis*. New York: Wiley. ISBN 0-471-20811-6.
11. Silicon Hive (2008). Silicon Hive homepage. http://www.siliconhive.com.
12. Texas Instruments (2008). TI MSP430F149 Datasheet. www.ti.com.
13. True System-on-Chip with Low Power RF Transceiver and 8051 MCU. TI Datasheet CC1110, SWRS033A.

Lennart Yseboodt currently works as researcher at Philips Research Eindhoven, working on low power sensor node architectures. He received his master's degree from the TU Eindhoven in 2007. His thesis was on a low power system level design for low power wireless sensor nodes. He's the author of a successful open source project: EFSL. His main inetrest lie in digital architectures and embedded software.

Michael De Nil currently works as researcher at IMEC Netherlands, working on ultra-low power DSP architectures. He received his master's degree from the TU Eindhoven in March 2007. His master thesis was on ultra-low power ASIP design for wireless sensor nodes, and was executed at Silicon Hive / IMEC.

Jos Huisken is currently principal researcher in IMEC Netherlands involved in the ultra low-power DSP activities for wireless autonomous transducer systems. After graduation in 1984 in University of Twente he joined Philips Reseach working in VLSI design and high-level synthesis for DSP. As user of architectural synthesis tools he created the first fully integrated VLSI design for Digital Audio Broadcasting (DAB) baseband processing and some IP blocks for the 3G mobile standard. The architecture template driven design approach led in 2002 to the creation of Silicon Hive, a spinout from Philips Research which he left in 2008 to join IMEC.

Mladen Berekovic has received the Dipl.-Ing. and Dr.-Ing degrees both from the University of Hannover, Germany, in electrical and computer engineering.

He has been Research Assistant with the Institute for Microelectronic Systems at the University of Hanover where he worked on several programmable video processor chips for MPEG-4 and image processing. After his PhD he worked at IBM on processor development and at IMEC as a senior researcher for advanced architectures. At IMEC he was leading teams on reconfigurable architectures and ultra-low-power DSPs. Prof. Berekovic was part-time assistant professor at the computer engineering department of Delft University of Technology, the Netherlands. Since 2007 he is professor at Technische Universität Braunschweig, Germany.

His present research interests include low-power VLSI implementations for signal processing, DSP and processor architectures for multimedia and wireless applications, reconfigurable and dependable computing systems.

Prof. Berekovic is a member of the IEEE and ACM, and served as a reviewer for several IEEE conferences and journal publications including DAC, IEEE Trans. CSVT, ETRI Journal and the Kluwer Journal of VLSI Signal Processing Systems. He is member of the Editorial Board of Elsevier's Journal on Microprocessors and Microsystems, and in the program committees of FPL, RAW, ARC, ARCS, SAMOS, Estimedia and VLSi-SOC.

Qin Zhao studied electrical engineering at Southeast University, Nanjing, China. She obtained her Ph.D degree in Information and Communication Systems Group at Eindhoven University of Technology. Her research topic includes code generation for embedded processors. Then she worked as software engineer at Takumi Technology B.V. for two years. In 2006, she joined Stichting IMEC Nederland as researcher. Her research interests are compiler technologies, code generation and system level design, etc.

Frank Bouwens received his Bachelor of Science from the Electrical System Engineering department at the Zeeland University of Professional Educational in 2003. He did his thesis at Philips Semiconductors, Nijmegen, the Netherland on embedded automotive system. He received a Master of Science degree in Computer Engineering at the Delft University of Technology in 2006. His Master thesis was in power and performance optimizations of the coarse-grained reconfigurable architecture *ADRES* at IMEC Leuven, Belgium. He is currently working at the Holst Centre for Stichting IMEC-NL as a Researcher on ultra low power solutions for DSPs. His research interests are energy efficient solutions for embedded medical systems and hardware/software co-design.

Jos Hulzink received his Master of Science degree in Electrical Engineering from the Eindhoven University of Technology in 2005. He graduated on a Master's Thesis about the optimization of Ultra Long Instruction Word processors for the Software Defined Radio (SDR) domain. Between 2005 and 2007 he worked as a SDR Application Engineer and Processor Designer for Silicon Hive. In 2007 he joined the Holst Centre (Eindhoven, the Netherlands) as a researcher Ultra Low Power DSPs, where he focuses on ultra low power, high performance processor architectures for SDR applications.

Jef van Meerbergen (M’87–SM’92) received the M.S. degree in electrical engineering and the Ph.D. degree from the Katholieke Universiteit Leuven, Belgium, in 1975 and 1980, respectively. In 1979, he joined Philips Research Eindhoven, Eindhoven, The Netherlands, where he started to design MOS digital circuits, domain-specific processors, and general-purpose digital signal processors. He was the Project Leader of the Sigma-Pi project which delivered the first general purpose DSP within Philips. In 1985, he started working on application-driven high-level synthesis in the context of a European project in close cooperation with Imec. Initially, this work was targeted toward DSP applications and resulted in the AR|T system which is used to design audio, video, and communication functions. Later, the application domain shifted toward high-throughput streaming applications for which the Phideo compiler was developed. This compiler was used for the design of feature box ICs for 100-Hz conversion for TV (Melzonic, Falconic) and for MPEG2 encoding (I.McIC). The Phideo paper received the best paper award at the 1997 ED&TC conference. His current interests are in design methods, heterogeneous multiprocessor systems, reconfigurable architectures and Networks-on-Silicon. He is a part-time professor at the Eindhoven University of Technology, Eindhoven, The Netherlands. Dr. van Meerbergen is a Philips Research Fellow and Associate Editor of Design Automation for Embedded Systems Journal.