Power model for wireless sensor node

van Emden, J.

Award date:
2007

TECHNISCHE UNIVERSITEIT EIDHOVEN
Department of Mathematics and Computer Science

Power model for wireless sensor node

By: Joris van Emden

Supervisors:
Jef van Meerbergen (TU/e)
Martijn Bennebroek (Philips research)

Eindhoven, October 2007

Contents

1 Introduction

This chapter starts with the problem statement which is followed by the explanation of Wireless Sensor Nodes (WSN) and Electrocardiogram (ECG). Next, the WSN architecture and power dissipation is discussed followed by possible optimizations of the WSN.

1.1 Problem statement

A big change is anticipated in modern healthcare from traditional care in hospitals toward remote care at home. The main drive is to counteract the ever increasing costs in the modern healthcare system. To enable this change, wireless body sensor nodes will be required that are able to monitor physiological parameters like ECG, blood pressure, oxygen saturation outside the hospital setting. These devices must be reliable, easy to use and have a long operation time. Ultimately one would like to develop devices that can operate without batteries enabling "unlimited" lifetime. Such low power devices pose a big challenge for the electronics because energy scavengers are only able to harvest 100µW whereas current systems consume a factor of 50 to 1000 more power.

In this study, the lower limit of energy-efficient device operation is investigated to check if a device is able to run on scavenged energy. For this, a model is created to give insight of the power usage of all system components. The model is first verified to a state-of-the-art System On Chip (SoC), this verification is done using all major system components. The system component will be exchanged to check their impact on total power usage and to locate bottlenecks in the device. Different applications will have diverse bottlenecks on power consumption while running on the same SoC. While one application may use the major part of the energy budget for computation, others may dissipate most for transmitting the data wirelessly. The model must be able to accommodate such application dependent features.

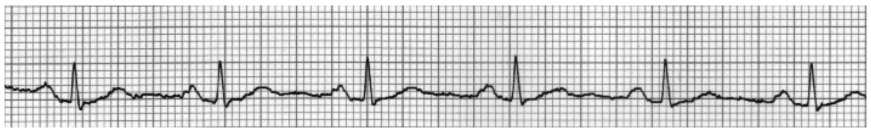


Figure 1.1: An ECG of a heart

This study focuses on electrocardiogram (ECG) applications to asses if it is possible to build a system able to run on scavenged energy. The results and underlying model can be readily reused to assess energy boundaries for other applications.

1.2 Wireless body sensors and ECG

The aging society lays a heavy burden on the economy nowadays. The number of patients needing care is increasing rapidly and this is overwhelming the the expensive care institution with work. To keep the cost of healtcare affordable, a shift from hospital-centric to remote care is needed. Remote healthcare needs Wireless Sensor Nodes (WSN) for remote monitoring and (in future) treatment. Next to long operation time and small dimensions, such WSN's must be extremely reliable and easy-to-use for (non-technical) patients and care givers. Examples of on-body worn monitors are temperature, blood pressure and ECG sensors. The latter will be discussed below because it is the focus of this study. Future treatment devices can also be placed inside the body, in this case artificial retina, pancreas and cochlear implants are made possible. In-body devices should work without batteries and use energy scavengers as power supply. The in-body WSN could save lives due to fewer operations needed for the patient. It could also change modern healthcare convenience to another level for patients, think for example about diabetic patients not needing to think about a shot of insulin after they have eaten or drunk something.

An ECG is the graphical representation of the electrical activity of the heart, as shown in Figure 1.1. It gives information about how the heart works and if any disorders are occurring. An important feature in the ECG is the QRS complex, seen in Figure 1.2, which shows the depolarization of the ventricles. The QRS is depicted in amplitude and duration. These parameters can be calculated by preparing the data first through (digital) filtering and beat detection, after which the QRS parameters can be easily extracted.

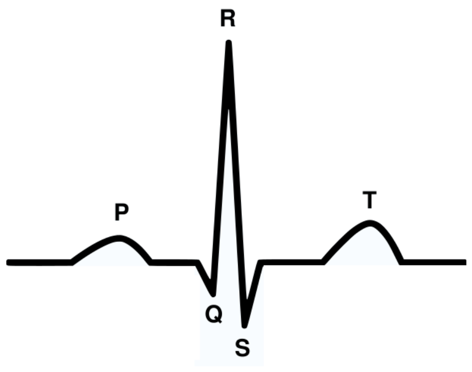


Figure 1.2: The QRS complex

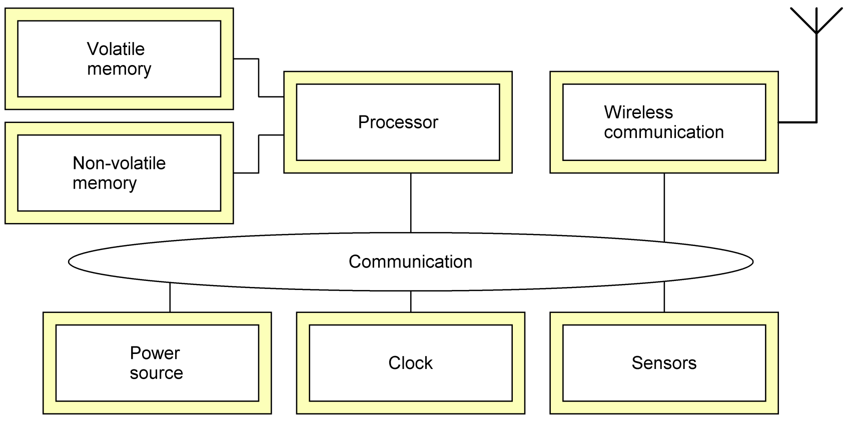


Figure 1.3: Architecture of a WSN

1.3 WSN architecture and power dissipation

This section describes the components from which a WSN is build. An architectural view is depicted in Figure 1.3 which shows the following components: processor with storage, wireless communication, sensors, real time clock generation, communication between the blocks and a power source. The storage is partly non-volatile memory to store the program and partly volatile memory for storing the data. State-of-the-art WSN SOC's, like the Chipcon 2430 (from now on is referred to as "Chipcon"), provide power modes in which one or more components are simultaneous active. To achieve low power consumption for a given application, efficient power modes needs to be enforced which basically aims to wake up only the required components while keeping all other components in deep sleep. The power consumption of component not being in deep sleep is unneeded. In a parallel study [8], low power operation at 5.60mW and 7.31mW is demonstrated on an ECG application build on a Chipcon 2430, transmitting RAW and QRS ECG data respectively. Figure 2.11 on page 33 depicts the underlying power usage which reveals wireless power dissipation can be relieved by sending local processed data instead of the RAW data.

1.4 WSN optimizations

This section gives insight into memory architectures and implementations, circuit implementation techniques and ends with the general expected result when an Integrated Circuit (IC) is scaled to another technology feature size. The most promising optimization techniques described in this section will be used to optimize the SoC in Chapter 3.

If feasible, battery less devices will undoubtedly have a huge potential. However, running without batteries has a big impact on the design of a system. Energy must be scavenged from the surrounding environment, implying that the system has very low power consumption. Current systems like the Chipcon 2430 are not capable to run solely on scavenged energy. Power optimizations need to be done to enable energy scavengers to completely power a WSN.

A lot of related research is currently done on battery- and wireless devices. However, these studies typically focus on one aspect of the device. In [3] for example, a Pearl processor is optimized for executing an ECG application, the software for this application is optimized in [4]. Other research focuses on energy scavengers [19], memory technology [16] [17] [18], radio technology (Philips), radio protocols [8], Analog to Digital Converters (ADC) [7] and clock generation [10]. The approach in the thesis is focused to get a global insight of the total power consumption of a complete SoC.

1.4.1 Memory architectures and implementations

Memory is needed to store the program being executed and program variables. We can distinguish different types of memory; volatile and non-volatile.

Non-volatile memory can retain the stored information even when it is not connected to a power source. This type of memory is normally used to store the program of the device because no power is needed to retain the data. Various types of non-volatile memory exist but all have some sort of disadvantage; Read Only Memory (ROM) can only be written once while FLASH can be rewritten but only a limited amount of times.

Volatile memories, like Random Access Memory (RAM) needs power to retain the contents. RAM has the advantage of an unlimited amount of read/write cycles, making it ideal to store data used by the executed algorithm.

Volatile memory

Filter cache
Large memories have a disadvantage over small memories; they consume more power and have a higher access time. Cache is used as an intermediate level because it uses less energy and time usage to access the data. Cache has copies of the data from the large memory. Disadvantage of the cache is that when the requested data is not available, time and energy is waisted in trying to get the data from the cache. The time penalty can be solved by setting up the regular memory transfer parallel with the cache, but only access the normal memory when the data is not available in the case. Solving the time penalty uses power and decreases the power savings of having a cache. The filter cache is a unusually small cache positioned between the core and the normal L1 memory of cache. A filter cache can save 58% power while the performance is reduced with 21%[12]. Other papers [13] show an average of 34% reduction in power while the performance reduction is less than 1%. On average, 15% of the requests to the filter cache result in a cache miss due to the small size. More about filter cashes can be found in [12], [13] and [14].

Actively reducing memory size
A way to save power in the data memories is to reduce the size of the memory, this is possible if it is not completely utilized. A memory can be constructed out of multiple memories while logically it looks like one big memory. Every small memory block of this logical big memory is power gated as explained in section 1.4.2. To use this technique the programmer of the system must be aware of how much and what part of the memory can be powered down.

The programmer of the system might not be able to fully utilize this technique. To solve this, new cache technologies in [16] [17] [18], in here they are investigated how to automatically disable memory parts which are not used.

Non-volatile memory

FLASH endurance problem
FLASH is ideal to store instructions for the processor because it is able to retain the data without a power source and can be reprogrammed. With low DC systems it is an interesting idea to use FLASH also as data storage because it can be disconnected from the power source to save leakage energy which is used in SRAM to maintain the data. The endurance (which means the number of cycles a block or chip can be reprogrammed) for FLASH is however limited. Typically FLASH has an endurance of one million[1] erase/write cycles which limit the usage of FLASH as data storage. With this endurance and a rewrite frequency of for instance 1Hz, the one millionth erase/write cycle limit is reached after 11.6 days.

FLASH might be used when the data is rewritten on a much slower frequency, but this is not feasible for a many applications. Other current technologies, like Ferroelectric Random Access Memory (FRAM), have a higher endurance (10 billion rewrite cycles[2]) but cannot be integrated into a SoC at the moment of writing and is not investigated further for this reason.

FLASH versus ROM
Non-volatile memory, like FLASH for instance, is needed to store the instructions of the device when it is not in use. A disadvantage of FLASH, being a high power consumption when active, can be solved by replacing it with ROM. The power usage of different memory sizes for FLASH and ROM are depicted in Figure 1.4. The drawback of ROM is that is can not be reprogrammed after fabrication. This limits the usage of ROM to high volume systems that are proven to work correctly. In this setting ROM has also another advantages over FLASH: it is a factor 2 to 5 times denser, reducing the cost of the device. Another advantage is, that ROM is available in the latest technology while FLASH is mostly lagging 2 feature sizes behind making it impossible to use the least power efficient technologies.

The advantage of being able to reprogram the FLASH will be traded to the lower power consumption of ROM in one optimization step executed in Chapter 3.

1.4.2 Circuit implementation techniques

Power gating

Power gating is a technique where the leakage of a (sub)component in a system is decreased when it is not active by removing the power supply to

[1]Source: Toshiba:www.dataio.com/pdf/NAND/Toshiba/NandDesignGuide.pdf.pdf
[2]Source: www.fujitsu.com/global/news/pr/archives/month/2007/20070418-02.html

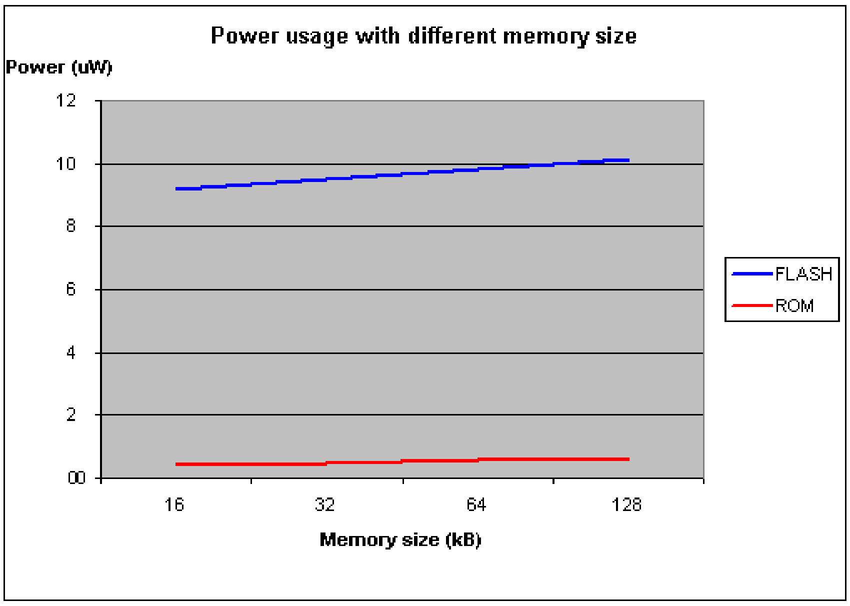


Figure 1.4: The power consumption for FLASH and ROM running on 32MHz with a duty cycle of 1%.

that component. An extra transistor is inserted between the power rail and the component to accomplish this, as shown in Figure 1.5. This technique can be applied for instance on the core of the processor when it is not active. External circuitry is needed to revive the processor. SRAM memories can also be power gated, however if done so, the contents will be lost when the supply voltage is removed.

The extra standby transistor will increase the active power consumption but this is neglectable with the power saved by this technique.

Threshold voltage of transistors: high V_t against normal V_t
The way how a transistor is made influences the threshold voltage. The threshold voltage describes the point where the transistor starts conducting.

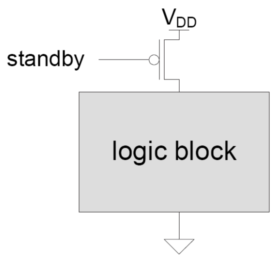


Figure 1.5: With the standby signal the logic block can be disconnected from the power rails, saving leakage when not used

A high threshold voltage will lead to less leakage when turned off, however it lowers the frequency at which the transistor is able to switch between states. If speed is not an issue, high V_t transistors should be used to save leakage.

To illustrate the difference in power consumption, a 8192 by 8-bit memory from [2] at 90nm is used. Both have the same active usage: 10pJ/access, while the leakage is 1.2µW and 11.3µW for high V_t and normal V_t respectively.

1.4.3 IC technology scaling trends

Energy consumption can be differentiated into static and dynamic. Static energy is the leakage of transistors, while dynamic energy is induced by the switching of transistors and (dis-)charging of the capacity in the circuit.

The technology depended feature size will influence the power consumption of digital circuitry. A smaller feature size will decrease the dynamic power (also known as active power), while the static power (also known as leakage) increases. Scaling the active power consumption between different technologies can can be calculated with the next formula:

$$\mathrm{P}_{\mathrm{active}}^{\mathrm{new}} = \left(\frac{\mathrm{V}_{\mathrm{dd}}^{\mathrm{new}}}{\mathrm{V}_{\mathrm{dd}}^{\mathrm{current}}}\right)^2 \cdot \frac{\mathrm{FeatureSize}_{\mathrm{new}}}{\mathrm{FeatureSize}_{\mathrm{current}}} \cdot \mathrm{P}_{\mathrm{active}}^{\mathrm{current}} \tag{1.1}$$

1.5 Summary

Power hungry WSN need to be optimized before they can be powered with scavenged energy. This thesis focuses on a global insight and optimization of a complete WSN SoC by means of a model The created model must be able to predict the power difference of the optimizations discussed in this chapter. The variety of optimization is huge; from technology implementation to component and architectural decisions. The most promising energy saving techniques, if applicable, are used to optimize the SoC in Chapter 3.

2 Model description

This chapter will describe how the WSN model is constructed. The model is parameterized in variables, the parameters together with the formulas explained in this chapter is the created model. The top view of the model, with all its subsystems, is depicted in Figure 2.3. The top view of the model will be explained before the explanation of the subsystem. The hardware of the WNS is divided into components which are specified by leakage, active energy and power consumption. In some cases, a duration of handling an event is added to the hardware description. The application will specify the activity of the components so that the energy consumption can be calculated.

The model is created in Microsoft Office Excel because this software package is widely used and known while it is still possible to expand the model for future use. It is build in a way that even non experienced Excel users can add new components. Excel offers the possibility to use Visual Basic for Applications (VBA), a programming language, so that even complex formula's can be composed. Another advantage of VBA is the possibility to add comments in the formula so that future users are able to understand what (and how) is calculated in the case they need to change or extend something.

The formulas used in this Chapter are constructed out of the next variables names followed by the explaination:

• P	Power
• E	Energy
• V	Voltage
• I	Current
• N	Amount
• T	Time

- τ Fraction
- f Frequency

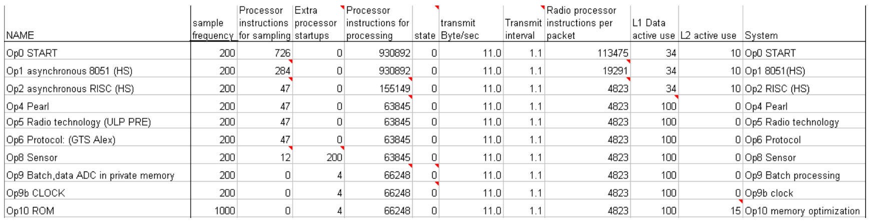
The rest of this section starts with explaining the application part of the model, followed by the explanation of the hardware components in the model. After the model is explained it will be trimmed in Section 2.3 to a System on Chip (SoC) to verify the model in Section 2.4. Fast readers go to the next chapter.

2.1 Application

The following list shows all application parameters to be set by the user:

- Amount of instruction for the application
- Amount of instruction for transmitting a radio packet (Medium Access Control, MAC)
- Amount of instruction for acquiring a single sample
- Amount of transmitted bytes per second
- Transmit interval
- Sampling frequency
- Extra high frequency crystal oscillator start up
- Level 1 Data memory activity
- Level 2 memory activity
- State save size

In Figure 2.1 the application parameters are named in the first row and the values for different systems are listen below that parameter. For instance, the "extra startups" apply to the high frequency crystal oscillator. This setting is made available to set extra startups of the processor to do application processing, which might not be triggered by a sampling or wireless communication event. The sampling frequency indicates obviously at what rate the sensor acquires data. Three different parameters can be given to state the amount of instructions the processor(s) has(have) to do, as indicated in Figure 2.1. The difference between them is when they are scheduled to execute. If the processor is needed to acquire a sample, for instance to copy the sample, it should be filled in at the amount of instructions needed to acquire one sample. The amount of instructions filled in for the sample will be multiplied with the sampling frequency of the sensor to come to the total



NAME	sample frequency	Processor instructions for sampling	Extra processor startups	Processor instructions for processing	state	transmit Byte/sec	Transmit interval	Radio processor instructions per packet	L1 Data active use	L2 active use	System
Op0 START	200	726	0	930892	0	11.0	1.1	113475	34	10	Op0 START
Op1 asynchronous 8051 (HS)	200	284	0	930892	0	11.0	1.1	19291	34	10	Op1 8051(HS)
Op2 asynchronous RISC (HS)	200	47	0	155149	0	11.0	1.1	4823	34	10	Op2 RISC (HS)
Op4 Pearl	200	47	0	63845	0	11.0	1.1	4823	100	0	Op4 Pearl
Op5 Radio technology (ULP PRE)	200	47	0	63845	0	11.0	1.1	4823	100	0	Op5 Radio technology
Op6 Protocol: (GTS Alex)	200	47	0	63845	0	11.0	1.1	4823	100	0	Op6 Protocol
Op8 Sensor	200	12	200	63845	0	11.0	1.1	4823	100	0	Op8 Sensor
Op9 Batch,data ADC in private memory	200	0	4	66248	0	11.0	1.1	4823	100	0	Op9 Batch processing
Op9b CLOCK	200	0	4	66248	0	11.0	1.1	4823	100	0	Op9b clock
Op10 ROM	1000	0	4	66248	0	11.0	1.1	4823	100	15	Op10 memory optimization

Figure 2.1: Snapshot of the input application worksheet

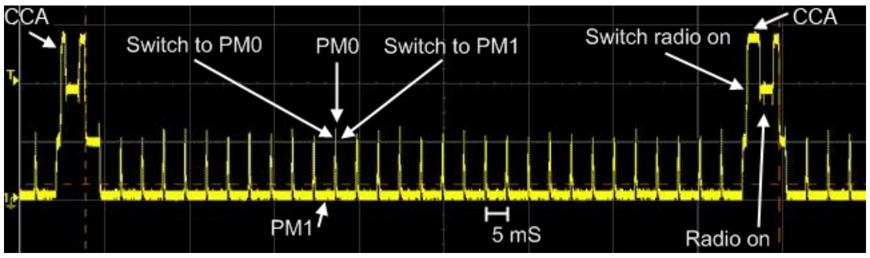


Figure 2.2: Current consumption of Chipcon in the "RAW" ECG application. Here, the device switches every 5ms from sleep mode (PM1) to active mode (PM0) to acquire a sample and wirelessly transmits groups of 32 samples

amount of executed instructions for sampling. The amount of instructions for radio processing will be multiplied with the amount of transmitted packets (called transmit interval) and executed on the processor selected by the radio (which can be another than the processor used for the application). All other instructions needed to be executed can be put at the amount of instructions for the application.

The usage of the different memories in the system depends on the application and must therefore be set in the model, a typical memory usage for data is 30%. The instruction memory is fully utilized for most processors and therefore it is automatically set to 100% active.

The state save size it the amount of data needed to be saved to L2 when the processor is power gated, how this works is explained further in Section 2.2.1.

Two parameters are used to describe the application side of the radio, transmit Byte/sec specifies how much data will be send wireless per second, while transmit interval specifies in how many packets per second this data will be put. The division in these two characteristics is needed to enable the packet rate to be easily changed without recalculating the amount of data transmitted per packet.

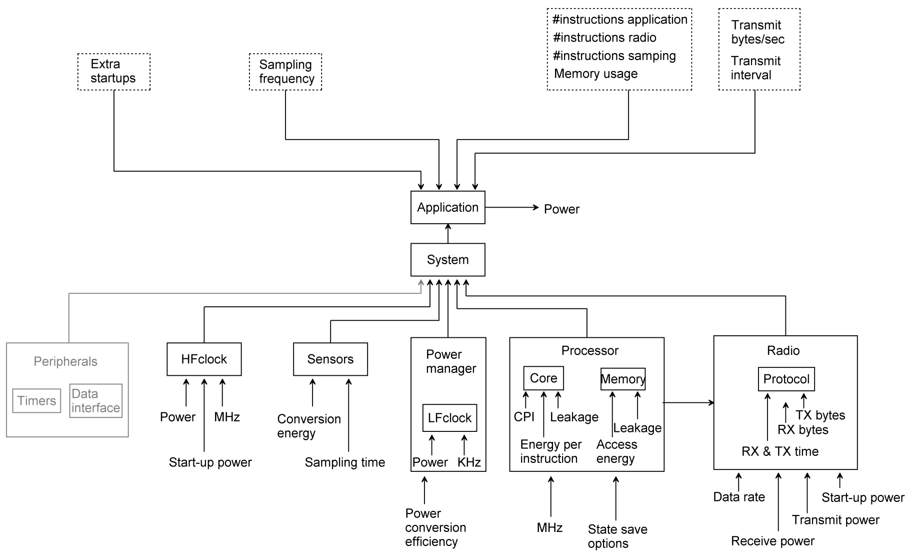


Figure 2.3: Model top view. Gray part are future extensions

2.2 System model

In Figure 2.2 the power consumption of a typical WSN is depicted, which the model must be able to predict. In this figure the power consumption of the Chipcon [1] is depicted when it executes the RAW ECG algorithm. The algorithm acquires samples that will be send by a radio. Figure 2.2 shows the different power consumptions of the Chipcon; PM1 where the processor is in sleep mode and PM0 where the processor is active.

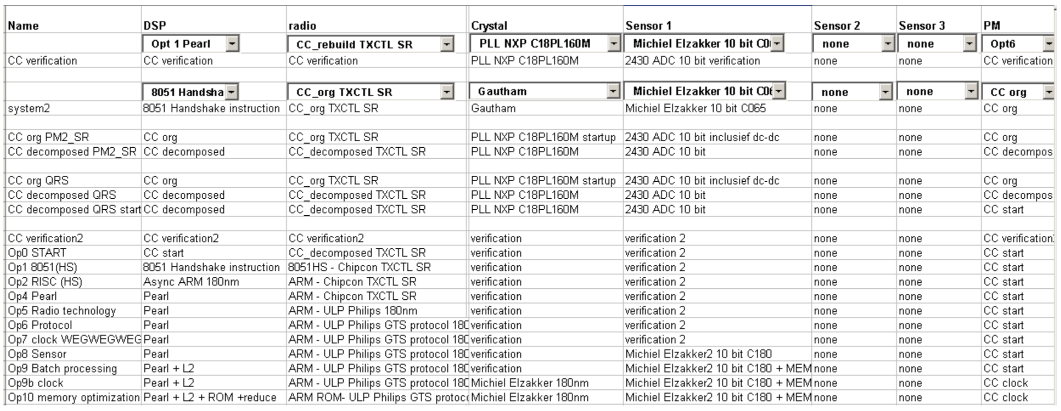
The power consumption of the complete system will be modeled by the summation of the different system components in Formula 2.1.

$$\mathrm{P_{total}} = \mathrm{P_{sensor}} + \mathrm{P_{radio}} + \mathrm{P_{processor}} + \mathrm{P_{PM}} + \mathrm{P_{HF}} + \mathrm{P_{peripheral}} \quad (2.1)$$

The power of the different components used in Formula 2.1 can be decomposed into a summation of the different power modes of the component and the summation of the power used to switch between these power modes, as depicted in Formula 2.2. These two formulas can calculate the power used in Figure 2.2.

$$\begin{aligned}\mathrm{P_{component}} &= \mathrm{P_{mode0}} + f_{\text{to \& from mode0}} \cdot \mathrm{E_{switch}^{to\ and\ from\ mode0}} \\ &+ \mathrm{P_{mode1}} + f_{\text{to \& from mode1}} \cdot \mathrm{E_{switch}^{to\ and\ from\ mode1}} \qquad (2.2)\end{aligned}$$

Figure 2.3 shows that the system is constructed out of components, which are subsequently divided in even smaller blocks. Take for instance the processor; it is constructed out of a core (which does the logical calculations) and memories(used to store the program and data). From Figure 2.3 it can



Name	DSP	radio	Crystal	Sensor 1	Sensor 2	Sensor 3	PM
	Opt 1 Pearl	CC_rebuild TXCTL SR	PLL NXP C18PL160M	Michiel Elzakker 10 bit C0	none	none	Opt6
CC verification	CC verification	CC verification	PLL NXP C18PL160M	2430 ADC 10 bit verification	none	none	CC verification
	8051 Handsha	CC_org TXCTL SR	Gautham	Michiel Elzakker 10 bit C0	none	none	CC org
system2	8051 Handshake instruction	CC_org TXCTL SR	Gautham	Michiel Elzakker 10 bit C065	none	none	CC org
CC org PM2_SR	CC org	CC_org TXCTL SR	PLL NXP C18PL160M startup	2430 ADC 10 bit inclusief dc-dc	none	none	CC org
CC decomposed PM2_SR	CC decomposed	CC_decomposed TXCTL SR	PLL NXP C18PL160M	2430 ADC 10 bit	none	none	CC decompos
CC org QRS	CC org	CC_org TXCTL SR	PLL NXP C18PL160M startup	2430 ADC 10 bit inclusief dc-dc	none	none	CC org
CC decomposed QRS	CC decomposed	CC_decomposed TXCTL SR	PLL NXP C18PL160M	2430 ADC 10 bit	none	none	CC decompos
CC decomposed QRS start	CC decomposed	CC_decomposed TXCTL SR	PLL NXP C18PL160M	2430 ADC 10 bit	none	none	CC start
CC verification2	CC verification2	CC verification2	verification	verification 2	none	none	CC verification
Op0 START	CC start	CC_decomposed TXCTL SR	verification	verification 2	none	none	CC start
Op1 8051(HS)	8051 Handshake instruction	8051HS - Chipcon TXCTL SR	verification	verification 2	none	none	CC start
Op2 RISC (HS)	Async ARM 180nm	ARM - Chipcon TXCTL SR	verification	verification 2	none	none	CC start
Op4 Pearl	Pearl	ARM - Chipcon TXCTL SR	verification	verification 2	none	none	CC start
Op5 Radio technology	Pearl	ARM - ULP Philips 180nm	verification	verification 2	none	none	CC start
Op6 Protocol	Pearl	ARM - ULP Philips GTS protocol 18	verification	verification 2	none	none	CC start
Op7 clock WEGWEGWEG	Pearl	ARM - ULP Philips GTS protocol 18	verification	verification 2	none	none	CC start
Op8 Sensor	Pearl	ARM - ULP Philips GTS protocol 18	verification	Michiel Elzakker2 10 bit C180	none	none	CC start
Op9 Batch processing	Pearl + L2	ARM - ULP Philips GTS protocol 18	verification	Michiel Elzakker2 10 bit C180 + MEM	none	none	CC start
Op9b clock	Pearl + L2	ARM - ULP Philips GTS protocol 18	Michiel Elzakker 180nm	Michiel Elzakker2 10 bit C180 + MEM	none	none	CC clock
Op10 memory optimization	Pearl + L2 + ROM +reduce	ARM ROM- ULP Philips GTS protoc	Michiel Elzakker 180nm	Michiel Elzakker2 10 bit C180 + MEM	none	none	CC clock

Figure 2.4: Snapshot of the input system worksheet

also be seen that the radio uses a processor as a component. A processor is also used for the radio, this may or may not be the same processor used for application processing. The model enables to construct a system bottom-up by selecting subcomponents but also top-down by selecting already defined components in the system worksheet as shown in Figure 2.4. Every row in the worksheet is a complete system and all systems shown in the figure are used in chapter 3. Pull-down lists are available for the first 2 systems to enable fast top-down selection of components. In Figure 2.3 only one sensor is shown, the model is actually prepared to use different sensors, as depicted in Figure 2.4 with sensor1, sensor2 and sensor3. The model is made to be easily extended to multiple sensors (>3). Using these parameter together with the application specific parameters will enable teh model to predict the power usage.

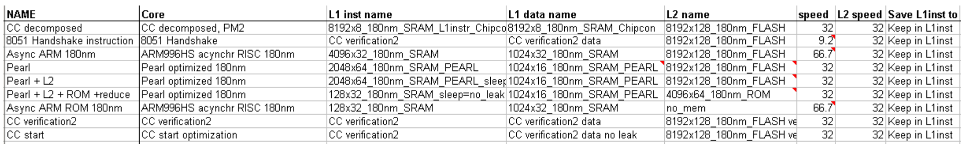
In the coming sections the input parameters stated in Figure 2.3 will be explained in detail and the separate components in Formula 2.1 will be decomposed to parameters that are used in the model.

2.2.1 Processor

In this section it is first explained from what parts the processor is composed and which options exist to change the power behavior of the processor. With this basic knowledge the formula for the power consumption of the processor will be explained.

The processor is the main computing part of the system and can be used to fulfill many different tasks in the system. It can, for instance, copy data from a sensor or compute features from the already gathered (sensor) information.

The core will be specified by the average energy used per instruction, the leakage power and the cycles per instructions (CPI). The CPI specifies the average amount of clock cycles needed for a core to execute an instruction.



NAME	Core	L1 inst name	L1 data name	L2 name	speed	L2 speed	Save L1inst to
CC decomposed	CC decomposed, PM2	8192x8_180nm_SRAM_L1instr_Chipco	8192x8_180nm_SRAM_Chipcon	8192x128_180nm_FLASH	32	32	Keep in L1inst
8051 Handshake instruction	8051 Handshake	CC verification2	CC verification2 data	8192x128_180nm_FLASH	9.2	32	Keep in L1inst
Async ARM 180nm	ARM996HS acynchr RISC 180nm	4096x32_180nm_SRAM	1024x32_180nm_SRAM	8192x128_180nm_FLASH	66.7	32	Keep in L1inst
Pearl	Pearl optimized 180nm	2048x64_180nm_SRAM_PEARL	1024x16_180nm_SRAM_PEARL	8192x128_180nm_FLASH	32	32	Keep in L1inst
Pearl + L2	Pearl optimized 180nm	2048x64_180nm_SRAM_PEARL_sleep	1024x16_180nm_SRAM_PEARL	8192x128_180nm_FLASH	32	32	Keep in L1inst
Pearl + L2 + ROM +reduce	Pearl optimized 180nm	128x32_180nm_SRAM_sleep=no_leak	1024x16_180nm_SRAM_PEARL	4096x64_180nm_ROM	32	32	Keep in L1inst
Async ARM ROM 180nm	ARM996HS acynchr RISC 180nm	128x32_180nm_SRAM	1024x32_180nm_SRAM	no_mem	66.7	32	Keep in L1inst
CC verification2	CC verification2	CC verification2	CC verification2 data	8192x128_180nm_FLASH ve	32	32	Keep in L1inst
CC start	CC start optimization	CC verification2	CC verification2 data no leak	8192x128_180nm_FLASH ve	32	32	Keep in L1inst

Figure 2.5: Snapshot of the input processor worksheet

The CPI can be below 1, meaning that more than a single instruction can be executed within one clock cycle. This is possible if the core has multiple issue slots i.e.: a Very Long Instruction Word (VLIW) processor. The model is also able to work with asynchronous processors, these processors do not have a clock and therefore a CPI can not be specified. These processors commonly use a handshake protocol to synchronize for internal communication and may use a clock to communicate with external devices. To indicate the usage of an asynchronous processor in the model, the CPI must be set to zero and the frequency of the core must be set to the average amount of executed instructions per second (IPS).

The core alone, as already explained, is not enough to create a complete working processor. To create a processor the core has to be selected together with memories, as shown in Figure 2.5. The memories connected directly to the processor are called level 1 memories (L1) and there is also a possibility to connect a second level (L2) memory.

The L2 is optional and existence mainly depends on the size of the L1 or the power saving properties of a second level cache when used correctly. The memories can be selected from a pull down list, which refers to memories in the "Memory" worksheet, which is filled with embedded memory information from NXP [2]. The memories are characterized in the model by the energy needed for an access and by leakage power.

Next to the core and the memories some other options can be set for the processor: the frequency of the core (at this speed the L1 memories will also run), the frequency of the L2 memory and some state saving options, as depicted in Figure 2.5. When the processor is power gated, the core and/or memories are disconnected from the supply voltage, as explained in 1.4.2. No leakage of the core will be added to the power when the core is disconnected from the supply voltage. SRAM looses it content when it is disconnected from the power rail. To enable power gating for the memories the model is able to calculate the power associated with saving the contents of the L1 memories to the L2 memory when they are disconnected from the supply voltage. This saving can be set independently for instruction and data memory. The instruction memory could be erased if the data is still present in another memory, the complete memory must be saved if this is not the case. The L1 data memory holds the state of the processor, this does not need to be the complete size of the L1 data memory. The state size is given as an input and only this size needs to be saved to L2 before the memory is disconnected from the supply voltage.

A processor will use energy when switched from sleep mode to active mode, this can be specified as start-up energy of the processor.

The last option to set for the processor is the possibility to reduce the amount of active L2 in sleep mode. There are 2 possible options: the original memory is completely powered up, or only the part what is actually needed to store the data is powered up. No change is done to the memory when the original memory is used. Power gating on parts of the memory, as explained 1.4.1 will be applied when memory size needs to be reduced. The size of the memory that needs to be active to store the data is determined by if the L1instr needs to be saved and the amount in 'state' explained in the previous paragraph. The model itself will try to find a suitable memory size able to store the requested data.

With the basic knowledge of the processor the formulas for the processor can be explained. Formula 2.3 states how the total power of the processor is calculated. The sample frequency is multiplied with the amount of instructions, needed by the processor to acquire a sample, and energy per instruction. The processor needs to switch from a sleep state to the active state if there are any instruction needed by the processor to acquire a sample (N>0), this switching involves energy which is asses to every sample acquired. The extra instructions multiplied with the energy per instruction and the extra startups from the processor are added next. The amount of state saves, depending on the $f_{\text{sample}} + f^{\text{extra}}_{\text{switching}} + f_{\text{transmit interval}}$, is multiplied with the power used to save the state which is explained later on in this section. The leakage of the memories need to be split up between the sleep and active time because in the sleep time it could be the case that (a part of the) memories are power gated and thus do not leak. The leakage of the memories when the processor is in the sleep mode is determined by the state saving options α, β and δ (zero if state saving is applied and one if state saving is not applied) whereas the duration of the sleep mode is the time that the processor is not active. The reduction in size of L2 will determine the $\text{P}^{\text{leakage}}_{\text{L2}}$ when the processor is in sleep. Note that the power for processing the radio instructions is not in this formula. The power of the radio processor is incorporated into the radio itself.

$$\begin{aligned}
\mathrm{P}_{\text{processor}} &= f_{\text{sample}} \cdot (\mathrm{N}_{\text{instr}}^{\text{sample}} \cdot \mathrm{E}_{\text{instr}}^{\text{processor}} + (\text{if N>0}) \cdot \mathrm{E}_{\text{switching}}^{\text{powermodes}}) \\
&+ \mathrm{N}_{\text{instr}}^{\text{extra}} \cdot \mathrm{E}_{\text{instr}}^{\text{processor}} \\
&+ f_{\text{switching}}^{\text{extra}} \cdot \mathrm{E}_{\text{switching}}^{\text{powermodes}} \\
&+ (f_{\text{sample}} + f_{\text{switching}}^{\text{extra}} + f_{\text{transmit interval}}) \cdot \mathrm{P}_{\text{state save}} \\
&+ (1 - \frac{f_{\text{sample}} \cdot \mathrm{N}_{\text{instr}}^{\text{sample}} + \mathrm{N}_{\text{instr}}^{\text{extra}}) \cdot \mathrm{CPI}}{f_{\text{core}}}) \\
&\cdot (\alpha \cdot \mathrm{P}_{\text{leakage}}^{\text{core}} + \beta \cdot \mathrm{P}_{\text{L1instr}}^{\text{leakage}} + \delta \cdot \mathrm{P}_{\text{L1data}}^{\text{leakage}}) \\
&+ \mathrm{P}_{\text{L2}}^{\text{leakage}} \qquad (2.3)
\end{aligned}$$

In Formula 2.4 the energy per instruction for the processor when active is stated. This is the summation of the energy per instruction of the core and energy per access on the memories (together with the power used in the bus), where τ_{L1instr}, τ_{L1data} and τ_{L2} determine activity fraction of the memories. The leakage per instruction is determined by the time of an instruction multiplied with the leakage power. The core and L2 can have different frequencies so the leakage of the core, L1instr and L1 data are multiplied with frequency of the core, while the leakage of the L2 is multiplied with it own frequency. The time of one instruction must be compensated in the leakage of the memories, this is done by the CPI.

$$\begin{aligned}
\mathrm{E}_{\text{instr}}^{\text{processor}} &= \mathrm{E}_{\text{core}}^{\text{active}} + \tau_{\text{L1instr}} \cdot (\mathrm{E}_{\text{L1instr}}^{\text{active}} + \mathrm{E}_{\text{L1 instr bus}}^{\text{active}}) \\
&+ \tau_{\text{L1data}} \cdot (\mathrm{E}_{\text{L1data}}^{\text{active}} + \mathrm{E}_{\text{L1data bus}}^{\text{active}}) \\
&+ \tau_{\text{L2}} \cdot (\mathrm{E}_{\text{L2}}^{\text{active}} + \mathrm{E}_{\text{L2 bus}}^{\text{active}}) \\
&+ \frac{\mathrm{CPI}}{f_{\text{core}}} \cdot (\mathrm{P}_{\text{core}}^{\text{leak}} + \mathrm{P}_{\text{L1instr}}^{\text{leak}} + \mathrm{P}_{\text{L1data}}^{\text{leak}}) + \frac{\mathrm{CPI}}{f_{\text{L2}}} (\mathrm{P}_{\text{L2}}^{\text{leak}}) \qquad (2.4)
\end{aligned}$$

$$\begin{aligned}
\mathrm{P}_{\text{state save}} &= \beta \cdot \frac{\mathrm{N}_{\text{Bytes to save for L1instr}}}{\mathrm{N}_{\text{L2 wordsize in bytes}}} \\
&\cdot \left(\mathrm{E}_{\text{L2}}^{\text{active}} + \mathrm{E}_{\text{L1instr}}^{\text{active}} + \frac{f_{\text{processor}}}{f_{\text{L2}}} \cdot \frac{\mathrm{E}_{\text{core}}^{\text{active}} + \mathrm{E}_{\text{L1instr}}^{\text{active}}}{\mathrm{CPI}} \right) \\
&+ \delta \cdot \frac{\mathrm{N}_{\text{Bytes to save for L1data}}}{\mathrm{N}_{\text{L2 wordsize in bytes}}} \\
&\cdot \left(\mathrm{E}_{\text{L2}}^{\text{active}} + \mathrm{E}_{\text{L1data}}^{\text{active}} + \frac{f_{\text{processor}}}{f_{\text{L2}}} \cdot \frac{\mathrm{E}_{\text{core}}^{\text{active}} + \mathrm{E}_{\text{L1instr}}^{\text{active}}}{\mathrm{CPI}} \right) \qquad (2.5)
\end{aligned}$$

The energy spent for saving one state is depicted in Formula 2.5. This formula consists of two parts: one for the energy spent to save L1 instr and the other for L1data. This division is done because state saving can be selected independent for both memories. The energy usage for state saving the memories is calculated the roughly the same for both memories; first the amount of write cycles to the L2 is calculated by dividing the amount of bytes needed to be saved by the word size of L2. This amount of write cycles is multiplied with the power needed to do one such write cycle: energy for L2 write, energy for reading L1instr or L1data, and the energy spend by the processor while the data is written back. The frequency of the processor and L2 can be different, as a compensation the energy spent by the processor and L1instr must be multiplied by the term $\frac{f_{\text{processor}}}{f_{\text{L2}}}$. This can be done because it is assumed that $f_{\text{processor}} \leq f_{\text{L2}}$. The amount of instruction needed to save the state is not only dependent on the frequency difference of the processor and the L2 bus also on the CPI. Dividing by the CPI brings the amount instructions back to the real executed instructions in stead of the difference of the frequencies. The processor is assumed to be able to fill the L2 word size every time it is written to L2. For asynchronous processors, having a CPI of zero, the CPI will be set to $\frac{f_{\text{processor}}}{f_{\text{L2}}}$, canceling out the earlier frequency term, this can be done due to the previous assumption.

If the model is set to reduce the amount of L2 during state save, the model substitutes the selected L2 by a smaller memory if available. If fore instance only 2kB of data needs to be saved and the memory is larger, a smaller memory is selected automatically from the memory list. The substituted memory must fulfill the next requirements: created in the same technology and having the same data width. This reduction in memory size is in the line of reducing the memories as a optimization stated in Section 1.4.1.

2.2.2 Radio

The radio is composed out of the radio technology, protocol and radio processor. The radio technology specifies the sleep, idle, transmit and receive current together with the supply voltage, data rate of the radio and the startup energy. Radios need a high frequency input clock with a specific accuracy. These last two parameters can be put into the radio part of the model but are not used, they only are added for completeness.

$$\mathrm{P}_{\text{radio}} = f_{\text{packet}} \cdot \left(\mathrm{E}_{\text{protocol}} + \mathrm{E}_{\text{payload}} + \mathrm{E}_{\text{radio}}^{\text{processor}} + \mathrm{E}_{\text{radio}}^{\text{startup}}\right) + \mathrm{P}_{\text{radio}}^{\text{sleep}} \tag{2.6}$$

Formula 2.6 is used to determine the power used by the radio; the packet frequency is multiplies with the energy of one packet which in turn is constructed out of the energy for the protocol, payload, radio processor and radio startup. If the radio consumes power while inactive, a sleep power is added.

$$\begin{aligned}
\mathrm{E_{protocol}} &= (\mathrm{T_{protocol}} \cdot \tau^{\mathrm{RX}}_{\mathrm{protocol}} + \frac{\mathrm{N^{RX\ overhead}_{bytes}}}{\text{link speed}}) \cdot \mathrm{P^{RX}_{radio}} \\
&+ (\mathrm{T_{protocol}} \cdot \tau^{\mathrm{TX}}_{\mathrm{protocol}} + \frac{\mathrm{N^{TX\ overhead}_{bytes}}}{\text{link speed}}) \cdot \mathrm{P^{TX}_{radio}} \\
&+ \mathrm{T_{protocol}} \cdot (1 - (\tau^{\mathrm{TX}}_{\mathrm{protocol}} + \tau^{\mathrm{RX}}_{\mathrm{protocol}})) \cdot \mathrm{P^{idle}_{radio}} \qquad (2.7)
\end{aligned}$$

Protocols are used to specify how different wireless nodes communicate with each other. All protocols add a specific overhead to the data needed to be send, which is depicted in formula 2.7. The packet overhead is divided into three parts: receive, transmit and idle. The part of the protocol that is independent of the link speed is specified with a total protocol time ($\mathrm{T_{protocol}}$) for the tree parts. The time for the tree parts is calculated by multiplying the total protocol time with the fraction of that part. The receive and transmit part of the protocol also have a link speed dependent piece which is specified in amount of bytes per part. The time of every part is calculated before it is multiplied with the power used for that part.

$$\mathrm{E_{payload}} = \frac{\mathrm{N^{payload}_{bytes}}}{\text{link speed}} \cdot \mathrm{P^{TX}_{radio}} \qquad (2.8)$$

The power used to transmit the payload is calculated by the number of payload bytes divided by the link speed and multiplied with the transmit power.

$$\mathrm{E^{processor}_{radio}} = \mathrm{N^{processor}_{MAC\ instr}} \cdot \mathrm{E^{\ processor}_{instr}} \qquad (2.9)$$

For every transmitted packet a certain amount of instructions is needed by the radio processor. The $\mathrm{E^{processor}_{instr}}$ in Formula 2.9 is calculated by Formula 2.4 with the values of the radio processor, except for the power of the L2 memory. If the radio processor is a different processor than the application processor the L2 will be shared between the two processors. The L2 for the radio processor is ignored and the values of the application L2 processor are used for the radio processor.

2.2.3 Sensors

To characterize the sensors, active and sleep power, duration per event (in case of an ADC this is the time it takes to acquire one sample) and start-up energy should be given, as depicted in formula 2.10.

$$\mathrm{P_{sensor}} = f_{\mathrm{sensor}} \cdot (\mathrm{T^{duration}_{sensor}} \cdot \mathrm{P_{sensor}} + \mathrm{E^{startup}_{sensor}}) + \mathrm{P^{sleep}_{sensor}} \qquad (2.10)$$

f_{sensor} in formula 2.10 is an application dependent frequency and is not set in the hardware profile of the sensor. It is possible to enter the minimal required input clock. If the minimal input clock is set to 0, the high frequency crystal oscillator is not activated for the duration of the event, in all other cases the clock is started and continues to run for the duration of the event.

2.2.4 Peripherals

Peripherals in the system are components like timers, data interfaces (UART and IO-pads) or others. The peripherals are a future extension to the model, as depicted in Figure 2.3 in gray.

2.2.5 Clock

The clock generation can be specified supply voltage, active-(and) start-up ampere and startup time. The clock drives a clock net that consumes power which is modeled by the frequency of the clock multiplied with energy needed for one complete clock cycle. From these parameters the energy used by the high frequency oscillator can be calculated when the startup frequency and the Duty Cycle (DC, τ_{HF}) is known, as depicted in Formula 2.11. The accuracy is also given as an input for completeness, but not used in the calculations of the model.

$$\mathrm{P}_{HF} = f_{\text{startup}} \cdot \mathrm{E}_{\text{HF}}^{\text{startup}} + \tau_{\text{HF}} \cdot \left(\mathrm{P}_{\text{HF}}^{\text{active}} + f_{\text{clock}} \cdot \mathrm{E}_{\text{clock net}}^{\text{switch}}\right) \quad (2.11)$$

The startup frequency is the addition of three components: sensor, radio and extra startups. ξ, ψ and ζ for the sensor time are booleans specifying if the high frequency crystal oscillator needs to be active for sensor1, sensor2 or sensor3 respectively.

$$f_{\text{startup}} = (\xi \mathrm{OR} \psi \mathrm{OR} \zeta) \cdot f_{\text{sensor}} + f_{\text{radio}} + f_{\text{extra}} \quad (2.12)$$

The DC of the high frequency crystal oscillator is an addition of three factors: sensor time, radio time and extra processing time.

$$\begin{aligned} \tau_{\text{HF}} &= f_{\text{sensor}} \cdot \mathrm{MAX}\left[(\xi \cdot \mathrm{T}_{\text{sensor1}}), (\psi \cdot \mathrm{T}_{\text{sensor2}}), (\zeta \cdot \mathrm{T}_{\text{sensor3}}), \left(\frac{\mathrm{N}_{\text{instr}}^{\text{sensor}} \cdot \mathrm{CPI}}{f_{\text{processor}}}\right)\right] \\ &+ f_{\text{packets}} \cdot \mathrm{MAX}\left[\left(\frac{\mathrm{N}_{\text{instr}}^{\text{radio}} \cdot \mathrm{CPI}}{f_{\text{processor}}}\right), (\mathrm{T}_{\text{protocol}} + \mathrm{N}_{\text{bytes}}^{\text{payload}} \cdot \text{linkspeed})\right] \\ &+ \frac{\mathrm{CPI}}{f_{\text{processor}}} \cdot \mathrm{N}_{\text{instr}}^{\text{processor}} \end{aligned} \quad (2.13)$$

The sensor time is the maximum duration of the different sensors and the processor time. The maximum time of the processor and radio transmission is used to calculate the time that the high frequency crystal oscillator is active for the radio part. $\mathrm{T}_{\text{protocol}}$, $\mathrm{T}_{\text{packet}}^{\text{RX overhead}}$, $\mathrm{T}_{\text{packet}}^{\text{TX overhead}}$, $\mathrm{T}_{\text{packet}}^{\text{idle overhead}}$, $\mathrm{N}_{\text{bytes}}^{\text{TX overhead}}$, $\mathrm{N}_{\text{bytes}}^{\text{RX overhead}}$ and linkspeed depends on the same variables used in Formula 2.6. The last part in Formula 2.13 determines the DC used to execute the extra instructions by multiplying the time for one instruction with the amount of instructions.

2.2.6 Power manager

To stabilize and convert the input power supply to the correct voltage a DC:DC[1] converter is used. The efficiency of the converter can be put into the model, which is used to adjust all the power drawn in the system.

The power manager includes a continuously running crystal oscillator, this typically runs at a low frequency to save power and it is used to schedule global events in the system. The used crystal oscillator can be selected from the clock worksheet in the model. Therefore it has the same input parameters specified as in the section 2.2.5, however the startup power is not used because it is continuously running. Formula 2.14 states how to calculate the power for this crystal oscillator.

$$\mathrm{P_{LF}} = \mathrm{V_{oscillator}} \cdot \mathrm{I_{oscillator}} \tag{2.14}$$

Formula 2.1 on page 15 that states to total power used in the system is changed to Formula 2.15 due to the $\mathrm{P_{total}}$ being influenced by efficiency of the DC:DC converter. Next to the efficiency change, $\mathrm{P_{PM}}$ in Formula 2.1 is changed to LF in the new formula because it is the power drawn by the power manager.

$$\mathrm{P_{total}} = \frac{\mathrm{P_{sensor}} + \mathrm{P_{radio}} + \mathrm{P_{processor}} + \mathrm{P_{LF}} + \mathrm{P_{HF}} + \mathrm{P_{Peripheral}}}{\mathrm{efficiency}} \tag{2.15}$$

2.3 Model calibration on Chipcon 2430 SoC

In this section the instruction count of two applications, executed on the Chipcon, are determined. Next, the hardware parameters of the Chipcon components are calculated.

2.3.1 Estimation of instruction count

The first application is the RAW ECG application where two main functions are executed:

1. Acquiring ECG samples at 200Hz
2. Wireless transmission of groups of 32 sample values

Measurements on the Chipcon are done with an oscilloscope to determine the time needed to handle a job. The time can be determined very accurate because a led on the Chipcon platform can be toggled at the start and end of a function. The led can be monitored with one of the leads of the

[1]An on-chip DC:DC converter is mostly constructed out of an Low Drop-Out regulator

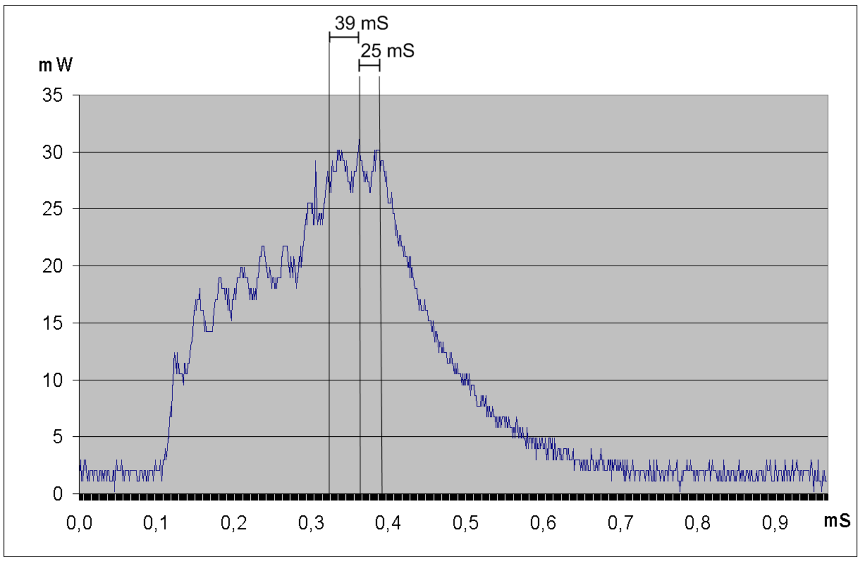


Figure 2.6: Measurement of taking a sample

oscilloscope. The measured time can be converted into instruction count by using the clock frequency and the CPI as illustrated in Formula 2.16.

$$N_{\text{instructions}} = T_{\text{measured}} \cdot \frac{f_{\text{HFoscillator}}}{\text{CPI}} \tag{2.16}$$

Figure 2.6 shows a measurement when taking a sample. In this figure, the Chipcon first has to wakeup from a low power mode before it can acquire a sample. The processor of the Chipcon is active for the duration of acquiring a sample (39µs), after which it needs 25µs to process the sample. The frequency of the Chipcon processor is 32MHz, and the CPI is 2.82 as determined in Appendix B, the amount of instructions per sample is determined to be 726 by using Formula 2.17. The 726 instructions is listed in the first row of Table 2.1.

$$\begin{aligned} N_{\text{instructions}} &= T_{\text{measured}} \cdot \frac{f_{\text{HFoscillator}}}{\text{CPI}} \\ &= (39\mu\text{s} + 25\mu\text{s}) \cdot \frac{32\text{MHz}}{2.82} = 726[1/\text{sample}] \end{aligned} \tag{2.17}$$

The measurement of a wireless transmission is shown in Figure 2.7. First the Chipcon does a part of the MAC processing in active mode (PM0) and then activates the radio in the receive mode (RX), followed by transmitting data (TX) and receiving the acknowledge (RX). After a packet is transmitted the processor stays in PM0 until the next sample is taken. It is needed to stay in PM0 because when a power down sequence on the Chipcon is activated (after a packet has been completely transmitted) and the wake-up timer is

	Sampling instr/sample	QRS extract instr/s	MAC instr/packet	Total instructions[a] instr/s
RAW ECG	726	-	113475	854419
QRS ECG	726	930892	113475	1200915

Table 2.1: Summary of instruction count per function and second

[a]With a sampling rate of 200Hz and a sampling rate of 6.25Hz for RAW ECG and 1.1Hz for QRS ECG

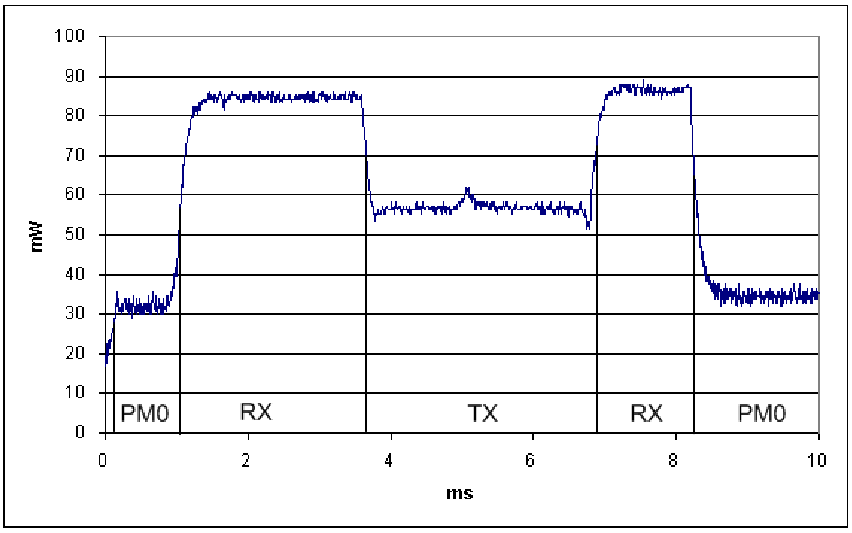


Figure 2.7: Power usage for transmission of a wireless packet

triggered (to acquire a new sample), the Chipcon goes to an undefined state where it stops responding. This causes the Chipcon to stay active for the wireless transmission in multiples of 5ms (which corresponds to the period of 200Hz sampling rate). The wireless transmission takes roughly 8ms, so the processor is active for 10ms. The amount of instructions used to transmit a single packet, as stated in Table 2.1, is calculated by formula 2.18.

$$\begin{aligned} \mathrm{N_{instructions}} &= \mathrm{T_{measured}} \cdot \frac{f_{\mathrm{HFoscillator}}}{\mathrm{CPI}} \\ &= 10\mathrm{ms} \cdot \frac{32\mathrm{MHz}}{2.82} = 113475[1/\mathrm{packet}] \qquad (2.18) \end{aligned}$$

The total amount of instructions per second for the RAW ECG application is calculated by formula 2.19 and listed in Table 2.1.

$$\begin{aligned} \mathrm{N^{per\ second}_{instructions}} &= f_{\mathrm{sample}} \cdot \mathrm{N^{sample}_{instr}} + f_{\mathrm{packet}} \cdot \mathrm{N^{processor}_{MAC\ instr}} \qquad (2.19) \\ &= 200\mathrm{Hz} \cdot 726\mathrm{instr} + 6.25\mathrm{Hz} \cdot 113475\mathrm{instr} \\ &= 854419[1/\mathrm{second}] \end{aligned}$$

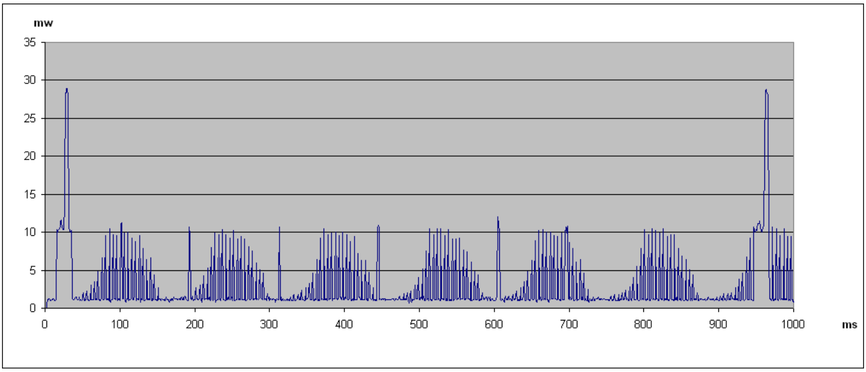


Figure 2.8: Power usage for transmission of a wireless packet

A sample frequency of 200Hz and transmitting packets containing 32 samples leads to a transmit interval of 6.25Hz. Since the application adds eight bytes per packet to the payload and every sample is represented in 2 bytes, the total amount of data send per second is: $6.25\text{packets/s} \cdot (32 \cdot 2\text{bytes} + 8\text{bytes}) = 450\text{bytes}$ per second which needs to be put in the model at the application worksheet under the name 'Amount of transmitted bytes per second'.

The second application is an ECG parameter extraction algorithm. Calculating the parameters is done by computing the QRS times of a heart beat, more details how the algorithm works can be found in [4]. The input of the algorithm are samples taken from the heart and the output is a 10 byte packet transmitted by the radio for every heart beat. The same amount of instruction are needed for acquiring the samples and transmitting a packet, which can be seen in Table 2.1.

A measurement of calculating the QRS parameters can be seen in Figure 2.8. The calculation of the QRS parameters involves three parts:

1. Filter the data at 200Hz
2. Beat detection at 4Hz
3. Calculate QRS parameters at heartbeat frequency

The measurement indicates that the filter function and beat detection takes 269µs and 2987µs respectively and the amount of instructions is calculated by formula 2.20 and 2.21.

$$\begin{aligned} \mathrm{N}_{\text{instructions}} &= \mathrm{T}_{\text{measured}} \cdot \frac{f_{\text{HFoscillator}}}{\text{CPI}} \\ &= 269\mu\text{s} \cdot \frac{32\text{MHz}}{2.82} = 3052[1/\text{sample}] \end{aligned} \tag{2.20}$$

$$\begin{aligned} \mathrm{N}_{\mathrm{instructions}} &= \mathrm{T}_{\mathrm{measured}} \cdot \frac{f_{\mathrm{HFoscillator}}}{\mathrm{CPI}} \\ &= 2987\mu\mathrm{s} \cdot \frac{32\mathrm{MHz}}{2.82} = 33895[1/250\mathrm{ms}] \end{aligned} \tag{2.21}$$

The duration of calculating the QRS parameters takes so long that three new samples are acquired and filtered in the mean time. The duration of acquiring and filtering these samples need to be subtracted from the measured duration of QRS parameter calculation. The measured time of the QRS parameter is 18.8ms, which besides the acquiring and filtering of the samples also includes the beat detection whih need to be subtracted. The amount of instructions is for calculating the QRS parameters is stated in Formula 2.22.

$$\begin{aligned} \mathrm{N}_{\mathrm{instructions}}^{\mathrm{QRS\ extract}} &= \mathrm{T}_{\mathrm{measured}} \cdot \frac{f_{\mathrm{HFoscillator}}}{\mathrm{CPI}} \\ &= \big(18.8\mathrm{ms} - (3 \cdot (39\mu\mathrm{s} + 25\mu\mathrm{s} + 269\mu\mathrm{s}) + 2987\mu\mathrm{s})\big) \\ &\cdot \frac{32\mathrm{MHz}}{2.82} = 168102[1/\mathrm{heart\ beat}] \end{aligned} \tag{2.22}$$

Knowing the average beat per minute, which is assumed to be 66bpm (=1.1Hz) it is possible to calculate the total executed instructions per second for the QRS parameters, as illustrated by Formula 2.23 and listed in Table 2.1.

$$\begin{aligned} \mathrm{N}_{\mathrm{instructions}}^{\mathrm{QRS\ total}} &= 200\mathrm{Hz} \cdot 3052 + 4\mathrm{Hz} \cdot 33895 + 1.1\mathrm{Hz} \cdot 168102 \\ &= 930892[1/\mathrm{second}] \end{aligned} \tag{2.23}$$

The total amount of instructions per second needed to do the QRS extraction is calculated in Formula 2.24, which is based on Formula 2.19 but an extra term is added for the extra instructions used for the QRS parameter extraction, and is also listed in Table 2.1, in this formula the packet rate is the same as the frequency of the heartbeat.

$$\begin{aligned} \mathrm{N}_{\mathrm{instructions}}^{\mathrm{total}} &= f_{\mathrm{sample}} \cdot \mathrm{N}_{\mathrm{instr}}^{\mathrm{sample}} + f_{\mathrm{instr}}^{\mathrm{extra}} \cdot \mathrm{N}_{\mathrm{instr}}^{\mathrm{QRS\ total}} + f_{\mathrm{packet}} \cdot \mathrm{N}_{\mathrm{instr}}^{\mathrm{MAC}} \\ &= 200\mathrm{Hz} \cdot 726 + 1\mathrm{Hz} \cdot 930892 + 1.1\mathrm{Hz} \cdot 113475 \\ &= 1200915[1/\mathrm{second}] \end{aligned} \tag{2.24}$$

Ten bytes are transmittes per packet. The frequency of the heartbeat is 1.1Hz and this is therefore also the packet frequency. The amount of bytes transmitted per second is $1.1\mathrm{Hz} \cdot 10\mathrm{bytes} = 11\mathrm{bytes}$.

2.3.2 Estimation of component power

The power consumption of individual Chipcon components is not listed in the Chipcon datasheet[1] and, unfortunately, can not be directly derived from measurements. In this section, literature, measurements and the Chipcon datasheet will be used to estimate the power of all components of the

Chipcon. The Chipcon has several Power Modes (PM), where PM0 is the active mode and PM1, PM2 and PM3 are sleep modes, where only parts of the Chipcon are active. PM0 and PM1 are measured to be 27.18mW and 1.05mW respectively. PM1 includes Power on reset (POR), sleep timer (ST), a kHz crystal oscillator, leakage of the digital part of the system including SRAM memories.

$$\begin{aligned} \mathrm{P}^{\mathrm{measured}}_{\mathrm{PM0}} \cdot \mathrm{LDO}_{\mathrm{Efficiency}} &= \mathrm{P}^{\mathrm{measured}}_{\mathrm{PM1}} \cdot \mathrm{LDO}_{\mathrm{Efficiency}} + \mathrm{P}^{\mathrm{32MHz}}_{\mathrm{clock\ net}} + \mathrm{P}_{\mathrm{32MHz\ Xosc}} \\ &+ \mathrm{P}^{\mathrm{active}}_{\mathrm{peripheral}} + \mathrm{P}^{\mathrm{FLASH}}_{\mathrm{Leakage}} + \mathrm{P}^{\mathrm{active}}_{\mathrm{mem}} + \mathrm{P}^{\mathrm{active}}_{\mathrm{core}} \end{aligned} \tag{2.25}$$

PM0 of the Chipcon equals the power drawn in PM1, 32MHz clock, the clock net, peripherals, FLASH leakage, power use in active memory and the power in the core. The measured PM0 and PM1 need to be corrected by the DC:DC efficiency before they can be used in Formula 2.25. The power consumption for all these parts will be determined.

The DC:DC converter, implemented as a Low Drop Out (LDO) regulator, has a conversion efficiency depicted in Formula 2.26, where the input voltage is 3V, the output voltage is 1.8V and the conversion efficiency θ is set to 100%.

$$\mathrm{LDO}_{\mathrm{Efficiency}} = \frac{\mathrm{V}_{\mathrm{out}}}{\mathrm{V}_{\mathrm{in}}} \cdot \theta \tag{2.26}$$

The power of the clock net can be calculated by Formula 2.27.

$$\mathrm{P}_{\mathrm{clock\ net}} = f \cdot \mathrm{C} \cdot \mathrm{V}^2 \tag{2.27}$$

The clock net is commonly constructed out of an H-tree, as illustrated in Figure 2.9, because it ensures an equal propagation delays to each end point of the tree. Every inner buffer in the H-tree powers 4 buffers toward the end of the tree. The total amount of buffers in the H-tree can be calculated by Formula 2.28, where L is the amount of levels in the tree. The amount of

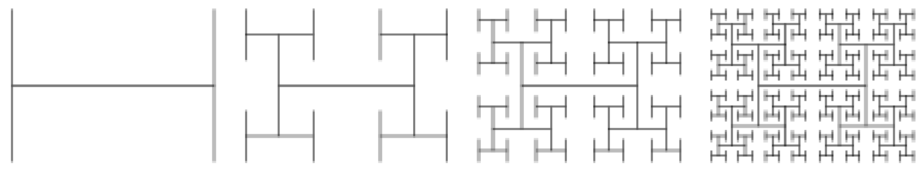


Figure 2.9: H-tree of levels two to five

end buffers in the H-tree is calculated with Formula 2.29.

$$\mathrm{N}^{\mathrm{total}}_{\mathrm{buffers}} = \sum_{0}^{L-1} 4^L \tag{2.28}$$

$$N_{\text{buffers}}^{\text{end}} = 4^{L-1} \tag{2.29}$$

It is assumed that there are 5 levels in the Chipcon clock net, bringing to total buffers in the clock net to 341. Every end buffer is assumed to drive eight Flip-flops and the average wire length per buffer and to the Flip-flops is 0.3mm. A input capacitance of a buffer and Flip-flop in 180nm is 4.7fF and the wire capacitance 100fF/mm, both values come from a NXP library. The total capacitance for the clock net is calculated by Formula 2.30 by multiplying the amount of gates with the capacity per gate.

$$\begin{aligned} C &= (341 + 256 \cdot 8) \cdot (4.7\text{fF} + 0.3\text{mm} \cdot 100\text{fF/mm}) \\ &= 82.9\text{pF} \end{aligned} \tag{2.30}$$

The total power of the $\text{P}_{\text{clock net}}$ can be calculated with Formula 2.27, which leads to 8.60mW with a clock frequency of 32MHz and a supply voltage of 1.8V.

Power consumed by the 32MHz crystal oscillator can be extracted from the Chipcon datasheet; the processor running with low activity on a 16MHz RC-oscillator consumes 7.74mW. With the same low activity but now on 32MHz it consumes 17.1mW, the frequency is generated with a crystal oscillator. The active part of the processor scales linear with the frequency, so if the static energy (indicated with PM1) is subtracted it is possible to calculate the power consumed by the crystal oscillator because the power used by the RC-oscillator can be neglected, as depicted in Formula 2.31.

$$\begin{aligned} \text{P}_{\text{32MHz Xosc}} &= \left(\text{P}_{\text{PM0}}^{\text{32MHz}} - \text{P}_{\text{PM1}}\right) \\ &- \left(\frac{32\text{MHz}}{16\text{MHz}} \cdot \left(\text{P}_{\text{PM0}}^{\text{16MHz}} - \text{P}_{\text{PM1}}\right)\right) \\ &= \left(17.1\text{mW} - 342\mu\text{W}\right) \\ &- \left(\frac{32\text{MHz}}{16\text{MHz}} \cdot \left(7.74\text{mW} - 342\mu\text{W}\right)\right) \\ &= 1.962\text{mW} \end{aligned} \tag{2.31}$$

This power consumption is inline with measurement performed in a similar manner that point to 2.5mW. Difference might be due to different devices.

$\text{P}_{\text{peripheral}}$ in Formula 2.25 is set to zero because it is assumed to be negligible.

From the Chipcon datasheet it is know that three different memories are used; two 4kB SRAMs, one 128kB FLASH and internal registers of the core, as illustrated in Figure 2.10. One 4kB SRAM is partially used in the register and the SFR register and it is unsure if it is also used for the instruction cache. The Chipcon datasheet states that both SRAM memories are build with high V_t memory technology. Using the memory estimator from NXP[2] the energy per access and leakage will be approximated for the memories

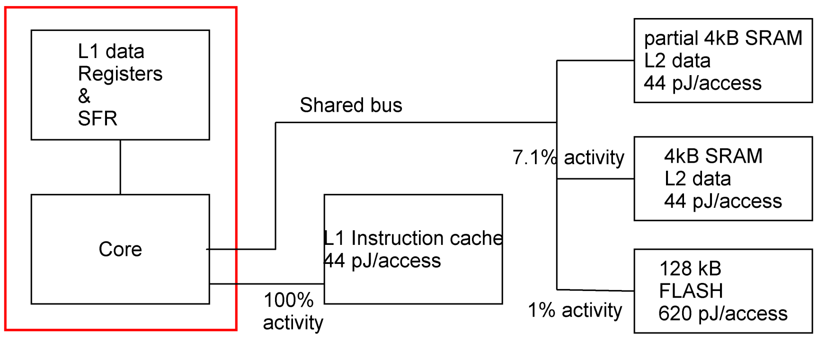


Figure 2.10: Chipcon core with memories and buses activity

interpreted in the Chipcon. A 4kB ultra low power SRAM uses 44pJ/access and leaks 180nW, embedded FLASH uses 620pJ/access and leaks 405µW. The leakage of both SRAM memories are already included into PM1 and are therefore already subtracted from the Chipcon power, the leakage of the FLASH is not included in PM1 and is directly subtracted in Formula 2.25 from PM0. FLASH is used to store the instruction for the processor, it is stated in the Chipcon datasheet that there is a cache between the FLASH and the processor. The size of this cache is unknown and it is assumed that it is byte addressable because the instruction width of the processor is one byte, further it is assumed that it has the same active power as a 4kB SRAM From [15] it is known that an typical instruction cache has a miss rate of 1%, which in term is used as the activity of the FLASH. The 8051 has internal registers and we assume that the power in these registers are included in the power of the core, as illustrated with a red box in Figure 2.10. The internal registers are L1 data for the core, the L2 data for the core is the 4kB SRAM. Because the registers are included in the core, the name of the L2 data in the model will be L1 data.

The power of the memories is calculated in Formula 2.32, which is partially derived from Formula 2.4: the amount of instructions per second multiplied with the power of the memories used for every instruction.

$$\begin{aligned} P_{MEM}^{active} &= \frac{\text{instruction}}{\text{second}} \cdot \big(\tau_{\text{instr cache}} \cdot (\mathrm{E}_{\text{instr cache}}/acc + \mathrm{E}_{\text{bus}}^{\text{instr cache}}) \\ &+ \tau_{\text{external data}} \cdot (\mathrm{E}_{\text{external data}}/\text{acc} + \mathrm{E}_{\text{bus}}^{\text{external data}}) \\ &+ \tau_{\text{FLASH}} \cdot (\mathrm{E}_{\text{FLASH}}/\text{acc} + \mathrm{E}_{\text{bus}}^{\text{FLASH}})\big) \end{aligned} \tag{2.32}$$

In most processors the activity of the L1instr is around 100% because every execution of an instruction involves fetching the instruction. One instruction fetch on the 8051 is 1 byte, but because some instructions of the 8051 uses multiple bytes for one instruction the activity will be more than 100 %. In appendix B it is shown that on average 1.46 bytes are used for an instruction, leading to a 146% activity to the L1instr cache. The data memory activity is determined to be 7.1% by counting the amount of MOVX instructions that refer to external memory divided by the total amount of instructions used for the filter code of the QRS application.

Formula 2.33 calculates the energy per bus transaction, which is the width of the bus multiplied with the capacity and squared voltage. Not all wires in a bus change per transaction, it is assumed that 2/3 of the wires actually changes state and thus consume power.

$$\mathrm{E}_{\text{bus}}^{\text{transaction}} = \mathrm{N}_{\text{bus width}} \cdot \frac{2}{3} \cdot \mathrm{C}_{\text{wire}} \cdot \mathrm{V}^2 \tag{2.33}$$

The capacity of a wire in Formula 2.33 can be calculated by multiplying the amount of gates per wire with the gate capacity and adding the capacity of the wire, as depicted in Formula 2.34.

$$\mathrm{C}_{\text{wire}} = \mathrm{N}_{\text{gates}}^{\text{per wire}} \cdot \mathrm{C}_{\text{gate}} + \mathrm{L}_{\text{wire}} \cdot \mathrm{C}_{\text{per L}} \tag{2.34}$$

The bus from the core to the instruction cache has a width of 8 bit and is a dedicated bus, meaning that it only has 2 gates per wire, assuming a bus length of 0.2mm the energy per bus transaction is 508fJ, calculated by using Formula 2.34 and 2.33. The bus to the FLASH and external data memory is shared, meaning that a lot of gates are connected to every wire. The width of the FLASH is 32-bit an therefore the bus has the same width. A bus transaction consumes 7.9pJ assuming a bus length of 0.2mm and 20 gates per wire, calculated by Formula 2.34 and 2.33.

All values are know to fill in Formula 2.32 to calculate the power used by the memories:

$$\begin{aligned} P_{MEM}^{active} &= \frac{32\text{MHz}}{2.82cpi} \cdot \big(1.46 \cdot (44\text{pJ} + 508\text{fJ}) \\ &+ 0.071 \cdot (44\text{pJ} + 7.9\text{pJ}) \\ &+ 0.01 \cdot (620\text{pJ} + 7.9\text{pJ})\big) \\ &= 850\mu\text{W} \end{aligned}$$

P_{core} can be calculated because all other term in Formula 2.25 are known, this leads to a 3.86mW. The power of the core can be converted in to energy per instruction by Formula 2.35.

$$\begin{aligned} \text{E/instr} &= P_{PROC}^{active} \cdot \frac{\text{CPI}}{f_{\text{core}}} \\ &= 3.86\text{mW} \cdot \frac{2.82}{32\text{MHz}} \\ &= 340\text{pJ/instr} \end{aligned} \tag{2.35}$$

As a reference, Handshake Solutions uses a reference 8051 consuming 500pJ/instr [2] which most likely involves a version not fully optimized for ultra low power.

[2]Source: Handshake Solutions

The start-up power is measured to be 6.3µJ. The radio power for both transmit and receive is measured to be 55.32mW, while the sleep power of the radio is zero due to power gating. The measured power should be adjusted with the DC:DC efficiency (Formula 2.26) before they are put in the model. The protocol times are derived from Figure 2.7 which the help of the work in [8], these times include switching between the radio modes and thus the power consumed for switching between the radio modes.

PM1 includes Power on reset (POR), sleep timer (ST), a kHz crystal oscillator, leakage of the digital part of the system and memory leakage due to data retention of both SRAM memories. Power involved in PM1 must be split up before it can be put in the model. The leakage of the SRAM memories is already covered in the memories itself. The kHz clock dissipation can not be measured because the power consumption is so low that is falls below the measure accuracy. It is assumed the the clock uses 396nW which is the power consumed by an EM microelectronic[9] crystal oscillator also running on 32kHz. POR power can be extracted from the Chipcon datasheet: 360nW. ST and peripherals are assumed to consume 500nW and 10.5µW respectively. The power of the digital leakage can than be calculated by subtracting all other parts from the PM1 power usage: $1.05\text{mW} \cdot \frac{1.8}{3} - \big(180\text{nW} + 180\text{nW} + 396\text{nW} + 360\text{nW} + 500\text{nW} + 10.5\mu\text{W}\big) = 617.9\text{mW}$. Note that the first two terms are the leakage the SRAM memories, which do need to be subtracted from PM1 to come to the leakage of the digital part.

The Chipcon datasheet specifies a consumption of 3.6mW for the ADC, this needs to be corrected with the DC:DC efficiency before it is put in the model.

2.4 Model Validation on Chipcon 2430 SoC

In Section 2.3.1 the amount of instructions is determined. The power usage of the Chipcon components is determined in a static way in Section 2.3.2. The model will estimate the dynamic power consumption of the Chipcon with the two above stated inputs.

The first application transmits the RAW ECG samples. The model predicts a power usage of 7.34mW, as illustrated in Figure 2.11. Measurement indicates that the Chipcon uses 7.31mW, a 0.4% difference to the model.

The second application that calculates the QRS parameters and transmits the QRS parameters wireless is predicted to use 5.52mW by the model, as illustrated in Figure 2.11 and was measured to use 5.60mW, which is a 1.4% difference.

A possible reason for the difference between the model and the measurement is the CCA time of the protocol. The CCA time is determined for every packet by a random variable, causing the measurement of the Chipcon

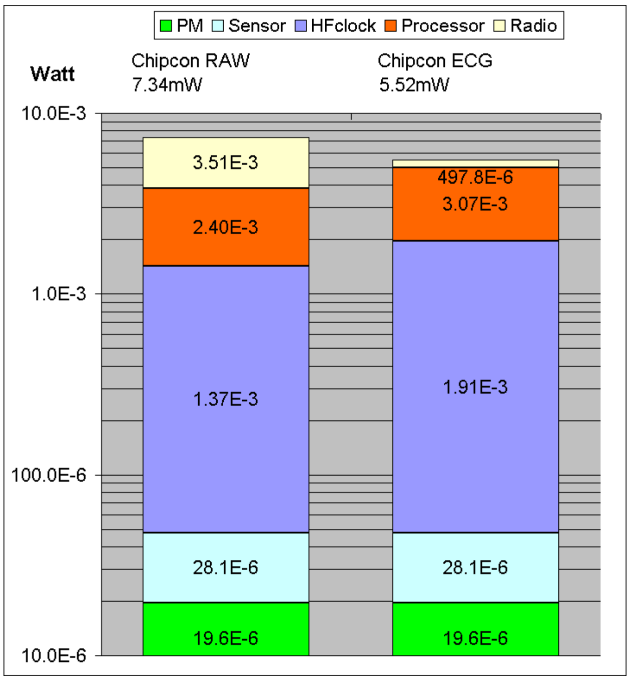


Figure 2.11: Power breakdown of RAW data and QRS parameter extraction. Note the logarithmic axis

to be non deterministic. The result of the random variable can be seen in Figure 2.2 on page 14. Two transmitted packets are illustrated in the figure, the first has a short CCA time, while the second has a long CCA time. The difference between the measurements and predictions of the model is small enough to state that the model works correctly.

From Figure 2.11 it is already possible to see that different applications have different power bottlenecks. This insight is made possible by the model due to dividing the power in components. Most power in the RAW mode is consumed by sending the data wireless By processing the data locally, as in the case of the QRS algorithm, more energy is evidently used for processing but this is compensated by bigger reduction in energy needed to transmit the data wireless. The next chapter will start with optimizing the biggest power consumer: the processor.

2.5 Summery

A complete description of the model is given in this chapter. The parameters used are described and all formulas in the model are explained. Two applications executed on the Chipcon are used to verify if the model works correctly. The verification showed that the model predict the power usage correctly, given that the input parameters in the model are correct.

3 System optimization results

This chapter will use the model described in chapter 2 to estimate the power of the various hardware optimizations. A reference system for executing the QRS parameter extraction algorithm, introduced in Section 2.3.1, will be the starting point of this chapter. The reference system will be optimized in consecutive steps to assess whether the ECG algorithm can run within the energy budget that allows for energy scavenging. The optimization exercises will be conducted assuming the entire system will be manufactured in 180nm CMOS technology. The optimization across process technologies is outside the scope of this study, although general formula's are stated in Section 1.4.3. Although a wide variety of optimizations scenario's have been considered in Chapter 1, a limited (the most promising) set will be used in this chapter.

3.1 Reference system

The QRS extraction application run on the Chipcon will remain the same, the hardware of the Chipcon will be changed. The unneeded peripherals are removed, which include, POR, UARTS, timers and IO ports. The majority of the power used in the peripherals goes to the IO ports, of the 21 IO pins available on the Chipcon, only one is needed for a serial interface or debugging purposes. This reduces the power consumed by the peripherals from 10.5µW to 0.5µW. The processor can be put into a deeper sleep mode (PM2) where the digital regulator is off, meaning that the core does not leak anymore, Beside this, only one of the two 4kB SRAM is powered in sleep mode, reducing the leakage of the memories by two in this mode. The transmit power of the radio can be adjusted, lowering the output power to from -0.1dBm to -25.4dBm leads to a power reduction in transmit mode from 39.9mW to 15.3mW.

The power waisted in the LDO is $1 - (\frac{1.8V}{3V}) = 40\%$. There are different options to get a better efficiency: decrease the input voltage or use a different

DC:DC converter with a higher efficiency. The latter option is chosen, meaning that the LDO is replaced with a switched capacitor converter. On-chip switched capacitor converters have a lower efficiency (67% according to [20]) compared to off-chip converters. Off-chip converters have a peak efficiency up to 90%, as illustrated in Figure 6.1 on page 56 which is taken from [21], and a broad operating range where the efficiency is above 85%. An off-chip switched capacitor converter is used in the system and it is assumed to have a 85% efficiency on the whole operating range.

All these optimizations leads to a power usage of 3.20mW for the reference syste, as illustrated in the first bar of Figure 3.1 on page 36.

3.2 Processor optimization

The power breakdown of the reference system, discussed in section 3.1, shows that the biggest power consumer is the processor. The processor is used to perform sampling, ECG processing and radio processing. This section describes how energy can be saved by removing the active wait instructions of the synchronous 8051 by using an asynchronous 8051 processor. The next optimization is to exchange the asynchronous 8051 with an asynchronous ARM to see the benefits of a processor with a better architecture. The last optimization is to use a second processor, especially optimized to execute the ECG application, next to the ARM.

3.2.1 Single processor design

The processor is used to fulfill 3 main tasks:

- Acquire a sample
- Extract the QRS parameters
- Arrange wireless communication

Acquiring the samples, as described in section 2.3.1, takes 726 instructions which is listed in Table 3.1.

The total sampling time is 64µs, 39µs of this is actually acquiring the sample and the remaining 25µW is for processing the sample. The 39µs the processor is executing wait instruction, so the actual needed instructions is $39\mu\text{W} \cdot \frac{32\text{MHz}}{2.82\text{CPI}} = 284$. The 284 instructions used for sampling is listed in Table 3.1 on the second row.

In Section 2.3.1 it is described that the wireless communication, involving the MAC processing and takes 113475 ($=10\text{ms} \cdot \frac{32\text{MHz}}{2.82\text{CPI}}$) instructions of which the majority are active wait instructions. A small test program has been created to check how many non-wait instructions are actually executed by the Chipcon to process the MAC. The test program used different timings

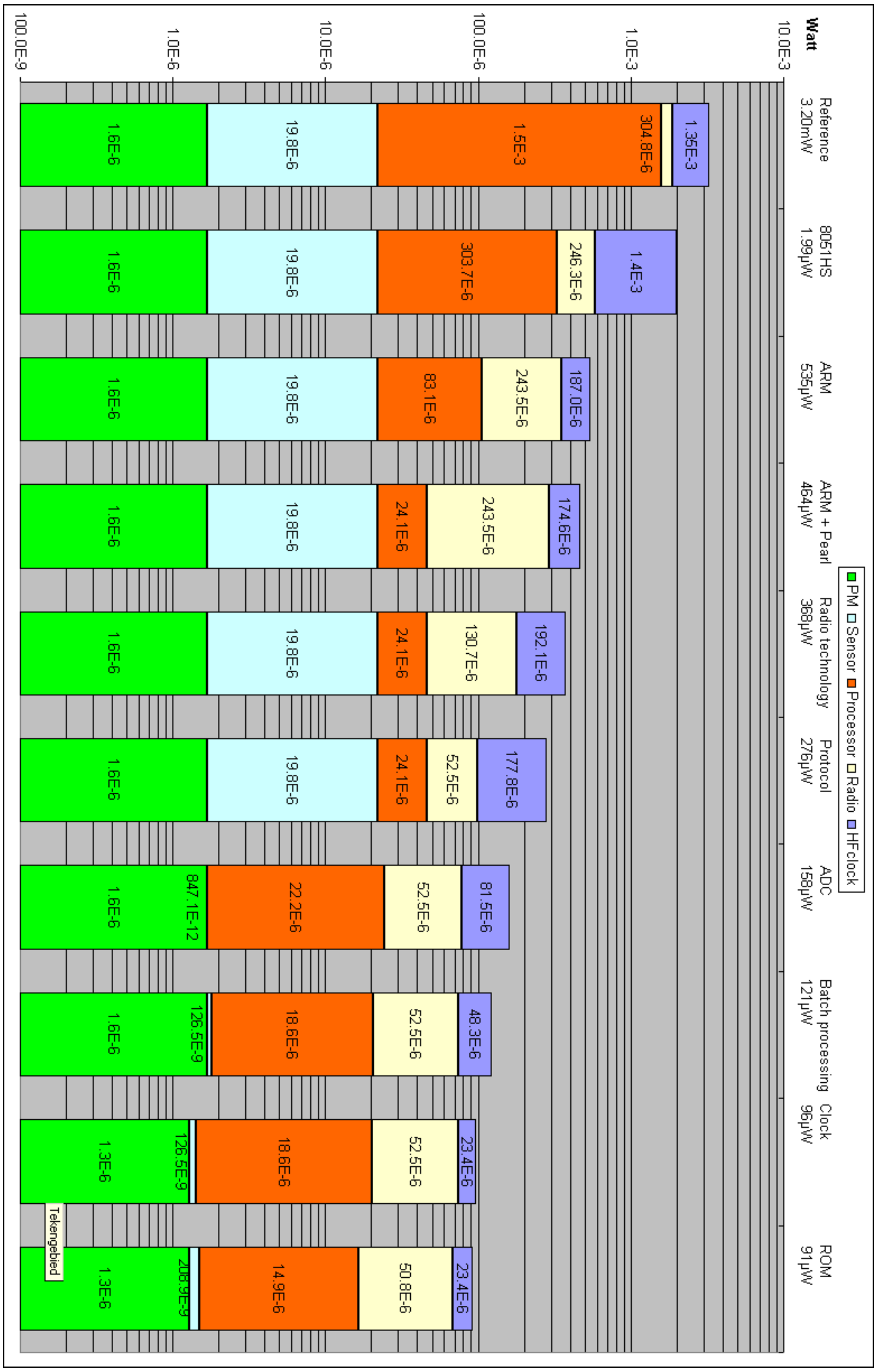


Figure 3.1: Power consumption breakdown of optimized ECG systems. In sub-bars that contain two values, the upper value reflects the power of the tiny sub-bar above. Note the logarithmic scale on the vertical axis

	Sampling instr/sample	QRS extract instr/s	MAC instr/packet	Energy per instruction J/instr
8051	726	930892	113475	340pJ/instr [a]
8051 HS	284	930892	19291	89pJ/instr [5]
ARM9 HS	47	155149	4823	214.8pJ/instr [b]
Pearl original		75406		50.83pJ/instr [c]
Pearl optimized	-	95766	-	39pJ/instr [3]

Table 3.1: Summary of instruction count per function and J/instr for the considered processors

[a]calculated in section 2.3.2

[b]Calculated using [6] by: 0.045mWMHz *1,5cpi/1MHz =67.5pJ/instr and scaled from 130nm to 180nm: 67.5pJ/instr $\cdot \frac{180\text{nm}}{130\text{nm}} \cdot \left(\frac{1.8\text{V}}{1.2\text{V}}\right)^2 = 214.8$pJ/instr

[c]It is 11.3pJ/instr at 90nm[3], 11.3pJ/instr $\cdot \frac{180\text{nm}}{90\text{nm}} \cdot (\frac{1.8\text{V}}{1.2\text{V}})^2$

in the protocol so that the time of the this protocol can not be compared to other protocols used in this thesis. The different protocol timing did not influence the amount of MAC instructions executed. A function is written that takes 252µs to complete, and is called continuously. Sending a packet takes around 5.6ms and handling the MAC processing was done by using interrupts, ensuring that the MAC processing was executed without interruption of the test function. When one packet is transmitted, the processor was able to complete 15 function calls. It is unknown to what extent the processor was able to finish the 16th call, so half the time for this was counted. The total time the processor was not utilized for MAC processing is $15.5 \cdot 252\mu\text{s} = 3.9\text{ms}$, the time needed to execute the MAC processing is therefore $5.6\text{ms} - 3.9\text{ms} = 1.7\text{ms}$. The processor executes $1.7\text{ms} \cdot \frac{32\text{MHz}}{2.82\text{CPI}} =$ 19291 instructions for the MAC, as listed in Table 3.1, all other instruction (113475 - 19291 = 94184) are active wait instructions and how these wait instructions are removed is explained next.

The execution of the QRS algorithm takes 930892 instruction per second, as described in section 2.3.1, there are no wait instructions involved in this part and therefore the amount of instructions stays the same as listed in the second row of Table 3.1.

Different techniques are available to reduce the amount of executed active wait instructions or to decrease the power of these instructions, like: processor halting, data holding, event handling (i.e. by interrupt request or asynchronous processor). It is chosen to use an asynchronous processor to eliminate the active wait instructions.

An asynchronous processor intrinsically avoids to be in an active wait state for acquiring a sample, nor should it be active after the MAC processing is completed for the wireless communication. This is true because the asynchronous processor is only active when an event enters the processor, the active wait instructions are eliminated by this. The synchronous 8051 processor is replaced by an asynchronous 8051 from Handshake Solutions[5] to remove the wait instructions.

The Handshake Solutions claims in [5] that the asynchronous 8051 uses 89pJ per instruction. The energy per instruction for the Chipcon core (340pJ/instr) is in line with the 500pJ/instr Handshake Solution used as a reference for a synchronous core. Handshake Solutions claims the decrease from 500pJ/instr for the 8051 reference to 89pJ/instr for the asynchronous 8051 due to the asynchronous design. In Section 2.3.2 it is assumed that the power of the registers in included into the core, this assumption is also used with the Handshake solutions core.

The memories in the system will remain the same because only the core is replaced, this means that the power used in the buses also remains the same because the distance from the core to the memories is equal.

A rough power save can be calculated by multiplying the amount of instructions from Formula 2.24 on page 27 with the energy per instruction for both processors, as depicted in Formula 3.1

$$\begin{aligned}
\frac{\mathrm{P}_{\mathrm{core}}^{8051}}{\mathrm{P}_{\mathrm{core}}^{8051\ \mathrm{HS}}} &= \frac{(f_{\mathrm{sample}} \cdot \mathrm{N}_{\mathrm{instr}}^{\mathrm{sample}} + f_{\mathrm{instructions}}^{\mathrm{extra}} \cdot \mathrm{N}_{\mathrm{instr}}^{\mathrm{extra}} + f_{\mathrm{packet}} \cdot \mathrm{N}_{\mathrm{MAC\ instr}}^{\mathrm{processor}}) \cdot \mathrm{E}_{\mathrm{core}}^{8051}}{(f_{\mathrm{sample}} \cdot \mathrm{N}_{\mathrm{instr}}^{\mathrm{sample}} + f_{\mathrm{instructions}}^{\mathrm{extra}} \cdot \mathrm{N}_{\mathrm{instr}}^{\mathrm{extra}} + f_{\mathrm{packet}} \cdot \mathrm{N}_{\mathrm{MAC\ instr}}^{\mathrm{processor}}) \cdot \mathrm{E}_{\mathrm{core}}^{8051\ \mathrm{HS}}} \quad (3.1) \\
&= \frac{(200\mathrm{Hz} \cdot 726 + 1\mathrm{Hz} \cdot 930892 + 1.1\mathrm{Hz} \cdot 113475) \cdot 430\mathrm{pJ/instr}}{(200\mathrm{Hz} \cdot 284 + 1\mathrm{Hz} \cdot 930892 + 1.1\mathrm{Hz} \cdot 19291) \cdot 89\mathrm{pJ/instr}} \\
&= 4.5
\end{aligned}$$

The gain expressed in Formula 3.1 comes from both the decreased instruction count and the more energy efficient processor.

The high frequency crystal oscillator is not needed for the core of the asynchronous processor but is still used for the memories attached to the core. The DC of the high frequency crystal oscillator, calculated by Formula 2.13 on page 22, is increased from 10.6% to 11.1% when the Handshake solutions core is used. This is caused by the lower instructions per second (denoted as MIPS: million instructions per second) the new core is able to process. The Chipcon core has a MIPS of: $\frac{32\mathrm{MHz}}{2.82\mathrm{CPI}}/1000000 = 11.3$ and the Handshake solutions core has a MIPS of 9.2[1]. The DC is divided into three parts: sampling, radio and calculation as depicted in Formula 22. The dominant part of the DC for the Chipcon is the calculation part (with 82ms it is 78% of the total DC). The time the crystal is on in the case of the Handshake Solutions core is: $\frac{11.3\mathrm{MIPS}}{9.2\mathrm{MIPS}} \cdot 82\mathrm{ms} = 101\mathrm{ms}$, which explains the increase. The increase in DC of the high frequency crystal oscillator also explains the increase in power for this component in Figure 3.1.

A more accurate estimate on power consumption for the whole system can be obtained by using Formula 2.3, 2.4, 2.6 and 2.13 as the model does,

[1]Taken from [5]

resulting in a power consumption of 303.7µW for the processor as depicted in the second bar from the left in figure 3.1.

After removing active wait instructions, significant energy savings can be obtained by using a more efficient processor. Appendix A explains the reduction of 6 times in instruction count used for data processing when the 8051 is replaced with an ARM processor. The six time reduction in instruction count is used in generating the instruction count for sampling and QRS extraction depicted in the third row of Table 3.1. MAC processing is mostly a control application and not a data processing application like sampling and QRS extraction. It is expected that a control application does not have the 6 time reduction in instruction count when the processor is changed from a 8051 to an ARM, instead a reduction of 4 times, as seen in Table 3.1 for the MAC processing is more suited for these kind of applications.

The replaced processor still has to be asynchronous to remove the active wait cycles as explained above. The selected processor is the ARM996[6] from Handshake Solutions; a clock-less 32-bit RISC processor core. The processor being 32-bit is actually a disadvantage because the program performs 16-bit computation and therefore can not make use of the 32-bits architecture, which results in an unneeded power usage in the wider data path of the processor.

The L1 instruction memory for the ARM has the same amount of lines (4096) for storing the instructions as the Chipcon, actually less lines are needed due to the decreased instruction count. The width of the instruction memory is changed from 8-bit to 32-bit to match the width of the processor, so the instruction memory of the ARM is 4096 words of 32-bit. The size of the L1 data memory is kept the same to 4kB, but the width is again changed from 8-bit to 32-bit, therefore the amount of parameter storage is reduced a factor four.

Energy used in the data bus stays the same because it was already a 32-bit bus in the previous optimizations. The energy in the instructions bus is recalculated with Formula 2.33 and 2.34 to be 2pJ per bus transaction.

Using the instruction count from Table 3.1 and Formula 3.1 the estimated power reduction is stated in Formula 3.2.

$$\begin{aligned}\frac{\mathrm{P}_{\mathrm{core}}^{\mathrm{8051HS}}}{\mathrm{P}_{\mathrm{core}}^{\mathrm{ARM\ HS}}} &= \frac{(200\mathrm{Hz}\cdot 284 + 1\mathrm{Hz}\cdot 930892 + 1.1\mathrm{Hz}\cdot 19291)\cdot 89\mathrm{pJ/instr}}{(200\mathrm{Hz}\cdot 47 + 1\mathrm{Hz}\cdot 155149 + 1.1\mathrm{Hz}\cdot 4823)\cdot 215\mathrm{pJ/instr}} \\ &= 2.46 \end{aligned} \tag{3.2}$$

The processor power with this optimization is reduced a factor 3.6 from 303.7µW to 83.1µW, as illustrated in Figure 3.1. The difference is caused by using different memories, bus power and other leakage of the processor.

Next to the drop in processor power in Figure 3.1 a drop in power for the high frequency crystal oscillator is illustrated too, the reason for this is twofold. First, the amount of instructions is decreased by roughly 6 times,

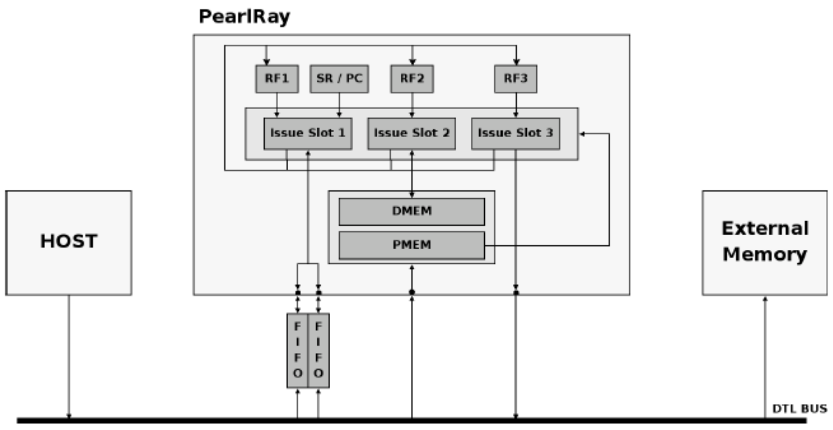


Figure 3.2: Original Pearl implementation

causing the oscillator to be active for a shorter time and second, the ARM is able to process more instructions per second: 66.7MIPS[2] for the ARM and 9.2MIPS for the 8051 from Handshake Solutions.

3.2.2 Dual processor design

An ultra low power solution for the processor can only be achieved by following an application specific approach; a general purpose core would never be as efficient as an application specific instruction set processor. In [3] [4] a Pearl processor from Silicon Hive is optimized for the ECG parameter extraction algorithm. The original Pearl is a 3 issue slot machine connected to a bus, see figure 3.2. Its hardware is described and altered in a high level programming language, the compiler is automatically generated to ensure the application code is properly mapped onto the changed hardware.

The Pearl is being optimized in [3] by several techniques: clock gating, memory size and memory technology for low leakage. The original data memory of 32kB is reduced to 2kB because the ECG application only needs 1.2kB of memory. The width of the instruction memory is reduced from 128-bit to 64-bit by removing the 3^{rd} issue slot and reducing the size of the immediates. As shown in Table 3.1, the original Pearl consumes 51pJ/instr, while the optimized version consumes 39pJ/instr (both converted by equation 1.1 at page 11 from 90nm to 180nm). The major part of the leakage is tackled by switching from normal V_t to high V_t transistors as explained in section 1.4.2.

From [4] it is known that all the optimizations result into a 27% increase in cycles of the program mainly because one issue slot was removed. This

[2]Calculated using [6]: reference ARM runs on 100MHz (and has equavalent performance) and a CPI of 1.5. 100MHz/1.5=66.7 million instruction per second

results into a 314 against the originally 248 cycles executed for filtering and detecting the beat in a sample, the QRS extraction takes 950 cycles. The total instruction count for the QRS extraction algorithm executed on the Pearl (with a CPI of 0.6661[3]) is: $\frac{200\text{Hz} \cdot 314\text{cycles} + 1.1\text{Hz} \cdot 950\text{cycles}}{0.6661\text{cpi}} = 95766$ instructions as listed in Table 3.1. A quick calculation from the same table learns that the original Pearl consumes $75406\text{instr} \cdot 50.83\text{pJ/instr} = 3.83\mu\text{J}$ on active energy, while the optimized Pearl consumes $95766 \cdot 39\text{pJ} = 3.73\mu\text{J}$. It does not look like a interesting gain, but the real gain in the optimized Pearl is within the idle and leakage power, detail information can be found in [3] and [4].

To put the memories of the Pearl the model it was possible to choose from the NXP library [2] or from the simulated memories [3] of the Pearl. The latter one was chosen because this is the actual power consumption which gives a more accurate result. The FLASH in the system is not used for the Pearl when it is executing instructions, due to the instruction memory of the Pearl being large enough to hold the complete program. The energy used in the buses, to let the Pearl communicate to the memories, are already included into the power simulations.

To give a fair comparison the frequency of the Pearl was set to 32MHz instead of the originally 100MHz it was build for in [3].

In previous sections the factor of active power save for the complete processor is calculated, in this case this is not done because only a fraction of the active power (only the QRS processing) is changed. The power saved in the part that has changed is calculated with Formula 3.3, which is a part of Formula 3.1, by subtracting the power used by the Pearl from the ARM and adjust it with the DC:DC efficiency

$$\begin{aligned} \frac{\text{P}^{\text{QRS}}_{\text{ARM}} - \text{P}^{\text{QRS}}_{\text{Pearl}}}{\text{DC}_{\text{efficiancy}}} &= \frac{(1\text{Hz} \cdot 155149 \cdot 214.8\text{pJ/instr}) - (1\text{Hz} \cdot 95766 \cdot 39\text{pJ/instr})}{0.85} \\ &= 34.8\mu\text{W} \qquad (3.3) \end{aligned}$$

The model, based on Formula 2.3, 2.4, 2.6 and others stated in Chapter 2, predicts a power save of (83.1µW - 24.1µW =) 59µW when the ARM processor is substituted by the Pearl to do the QRS algorithm. The difference between 59µW and 34.8µW is caused by not taking the different memories and leakages into account.

3.3 Radio optimization

After optimization of the processor described in the previous section, we now turn to the radio. Optimizing the radio is done in two manners: the radio technology and the protocol being used to transmit a packet.

[3]The CPI of the Pearl was determined by the work done in [3]

3.3.1 Radio Technology

Radio technology of Chipcon is rather power hungry, for example the receive stage consumes 33.2mW whereas the transmit stage consumes in the range of 15.3mW at -25.4dBm to 33.2mW at -0.1dBm[4]. Various academic and industrial activities are ongoing to develop radio solutions that are more energy efficient. For example, within Philips a radio is under development that provides a 50kb/s linkspeed up to 10meter at only 3mW transmit and receive power. The power of the new radio includes a PLL to transform the input of the high frequency crystal oscillator to the desired carrier frequency. The reduced range of 10 meters, compared to 75 meters for the Chipcon, is sufficient for the purpose of the system under investigation. With the data rate being 50kbps, it is one fifth of the 250kbps of the Chipcon radio. The decreased data rate is compensated by the lower power consumption. The estimated power when the radio is changed is stated by formula 3.4; by multiplying the changed linkspeed with the changed power consumption of the radio and the power used by the Chipcon radio.

$$\begin{aligned}
\Delta_{\text{linkspeed}} \cdot \Delta_{\text{Pradio}} \cdot \mathrm{P}^{\text{Chipcon}}_{\text{radio}} &= \frac{\text{linkspeed}^{\text{Philips radio}}_{\text{radio}}}{\text{linkspeed}^{\text{Chipcon Radio}}_{\text{radio}}} \cdot \frac{\mathrm{P}^{\text{average}}_{\text{Philips radio}}}{\mathrm{P}^{\text{average}}_{\text{Chipcon radio}}} \\
&\cdot\; \mathrm{P}^{\text{Chipcon}}_{\text{radio}} \qquad (3.4)\\
&= \frac{250\text{kb/s}}{50\text{kb/s}} \cdot \frac{3\text{mW}}{(15.3\text{mW} + 33.2\text{mW})/2} \\
&\cdot\; 243.5\mu\text{W} \\
&= 150.6\mu\text{W}
\end{aligned}$$

The model predict a power usage of 130.7µW, as depicted in figure 3.1, which is in line with the expected result. The difference can be explained by the included radio processing in the term $\mathrm{E}^{\text{processor}}_{\text{radio}}$ (in Formula 2.9) which does not scale with the power consumption of the new radio. Another factor explaining the difference is that the duty cycle of receive and transmit is not 50% as used in Formula 3.4.

3.3.2 Protocol

Selecting the optimum radio technology is not the complete story to optimize the radio; the overhead induced by the protocol to transmit the data can have a significant impact, specially if small packets are transmitted. The original protocol (CSMA) uses a Clear Channel Assessment (CCA) before transmitting a packet. This minimizes the collisions that might occur when multiple radios are using the same link frequency. Alternative protocols, like Guaranteed Time Slots (GTS), are available for low data rate networks.

[4]Maximum transmission power is 0.6dBm

GTS is a synchronized protocol where several nodes can have a predefined private slot. This makes it possible to send data without doing a CCA, however, the node must keep synchronized with the network coordinator. This synchronizing is done by using a beacon. Obviously some power is needed to receive the beacon but this is less compared to the power usage for CCA, because it is known when the beacon is send and the duration of it is less than the average CCA time, 2.9ms[5] and 8.4ms[6] respectively for GTS and CSMA with a linkspeed of 50 kbps. The private slot of GTS makes sure no collisions occur with nodes in the same network. More information about protocols can be found in [8].

The expected power when the protocol is changed from CSMA to GTS is calculated by Formula 3.5 by taking the changed protocol time multiplied with the power used for the radio.

$$\begin{aligned} \mathrm{P_{radio}} &= \frac{\mathrm{T^{GTS}_{protocol}}}{\mathrm{T^{CSMA}_{protocol}}} \cdot \mathrm{P^{previous}_{radio}} \qquad (3.5) \\ &= \frac{2.9\mathrm{ms}}{8.4\mathrm{ms}} \cdot 130.7\mathrm{mW} \\ &= 45.1\mathrm{mW} \end{aligned}$$

When the protocol is changed from CSMA to GTS the model predict a radio consumption of 52.5µW and not the estimated 45.1µW by Formula 3.5, the difference is caused by the protocol only being a part of the radio power, the other parts (payload and MAC processing) do not scale with the changed protocol.

The power usage of the radio is under the ideal assumption that no interference causing a needed retransmission of the data, the results of this assumption will be addressed in Section 4.1.

3.4 Sampling optimization

The highest power consumption of non-optimized components is the high frequency crystal oscillator: 177.8µW which is included in the HFclock part of Figure 3.1. The oscillator is replaced and the new oscillator has a lower continuous power consumption when active, but it has a higher total power consumption due to the increased energy needed to start the new oscillator. This optimization is rejected and the ADC is optimized in this section.

An unfortunate choice is being made in the Chipcon design that causes the sampling energy to be unnecessary high, one of these choices is that the high frequency crystal oscillator needs to be active while acquiring a sample. Next to this, the ADC has a high power consumption (2.16mW),

[5]Beacon time: 16 bytes $\cdot \frac{8}{50\mathrm{kbps}} + 320\mu\mathrm{s} = 2.9\mathrm{ms}$, data used from research done by [8]

[6]CCA time: 50 bytes $\cdot \frac{8}{50\mathrm{kbps}} + 400\mu\mathrm{s} = 8.4\mathrm{ms}$, data used from research done by [8]

a long acquiring time (39µs) and involves quite some processing due to the 10-bit value being stored unalligned in the 2-byte variable which leads to the need of shifting the data to the correct place of the two byte variable. By replacing the Chipcon ADC with a custom ADC of e.g. Philips[7], the above stated problems can be solved. The Philips ADC uses 7.2µW when active, has an acquiring time of 500ns and the data shift by the processor is not needed because it is already placed at the correct location in the variable. Processor instructions are saved by removing the need to shift the data; instead of the 47 instructions only 12 instructions are assumed to copy acquired sample to the memory. The power used to write the sample into the memory of the processor is already included into the energy per instruction of the processor: the activity of the data memory. The new ADC has an embedded RC oscillator so the high frequency crystal oscillator is not needed to acquire a sample. The power of the embedded RC oscillator is included into the power consumption of the sensor and the start-up time (and thus also the start-up power) is also included into the start-up time of the sensor.

Unfortunately the high frequency oscillator still needs to be started to let the processor copy each sample from the ADC to the memory.

The estimated power of the new sensor is calculated by using Formula 3.6, where we use the difference in power of the two sensors multiplied with the old sensor power

$$\begin{aligned} \Delta_{\mathrm{Psensor}} \cdot \mathrm{P}^{\mathrm{Chipcon}}_{\mathrm{sensor}} &= \frac{7.2\mu\mathrm{W} \cdot 500\mathrm{ns}}{2.16\mathrm{mW} \cdot 39\mu\mathrm{s}} \cdot 19.8\mu\mathrm{W} \\ &= 846\mathrm{pW} \end{aligned} \tag{3.6}$$

The model, as illustrated in Figure 3.1, predicts a power usage of 847.1pW, with is precisely the estimated value. Note in Figure 3.1 that the high frequency crystal oscillator uses less power when the sensor is optimized, the reason is twofold: first the ADC has an embedded RC oscillator so that the high frequency crystal oscillator is not active while acquiring a sample and second, the processor needs to execute less instructions per sample. The latter also explains the decreased power by the processor when this optimization is applied.

In principle, use of the HF oscillator can be circumvented by using the free running RC oscillator of the Chipcon (or from the sensor itself) for the non-timing critical sample to memory copy. Also, use of processor for sample to memory copy can be circumvented by using DMA. Both optimizations are not investigated.

3.5 Batch Program execution

The motivation for this optimization is to reduce the amount of startups for the high frequency crystal. Reducing the startups is possible if the processor is not copying data from the ADC to the memory sample-by-sample but for a burst of samples. For this, a memory is added to the ADC sensor to store the samples. An acceptable latency forces a batch frequency of minimal 4Hz, which in term leads to the sensor being able to store at least 50 samples due to the 200Hz sampling frequency. To store the 50 samples, a memory of 64 words of 16-bit wide is added to the sensor. The processor is started when the sensor memory is filled to copy the samples, after which the processor continuous to execute the ECG application. The processor instructions involved for copying the samples are moved from the sensor processing to the application processing because they are not executed when the ADC acquires a sample[7].

The expected power savings for this optimization is the reduction in power used by the high frequency crystal oscillator, however the increased power for the sensor caused by the added memory needs to taken into account too. The power of the sensor is calculate by formula 2.10 on page 21, $\mathrm{P}_{\mathrm{sensor}}^{\mathrm{sleep}}$ in this formula will be used to put the leakage of the memory ($50\mathrm{nA} \cdot 1.8\mathrm{V} = 90\mathrm{nW}$) into and $\mathrm{E}_{\mathrm{sensor}}^{\mathrm{startup}}$ will be used to add the active power, one for writing and one for reading the data, of the memory ($2 \cdot 42\mathrm{pJ} = 84\mathrm{pJ}$):

$$\begin{aligned} \mathrm{P}_{\mathrm{sensor}} &= 200\mathrm{Hz} \cdot (500\mathrm{ns} \cdot 4\mu\mathrm{W} + 84\mathrm{pJ}) + 90\mathrm{nW} \\ &= 107.5\mathrm{nW} \end{aligned}$$

The power of the high frequency crystal oscillator is reduced by the unneeded startups multiplied with the start-up energy:

$$(200\mathrm{Hz} - 5.1\mathrm{Hz}) \cdot 144\mathrm{nJ} = 28.1\mu\mathrm{W}$$

The power estimation in Figure 3.1 of the sensor is exactly the same if the DC:DC conversion efficiency is taken into account: $107.5\mathrm{nW}/85\% = 126.5\mathrm{nW}$. The power drop of the high frequency crystal oscillator is also the same if the DC:DC conversion efficiency is taken into account $28.1\mathrm{nW}/85\% = 33.0\mu\mathrm{W}$.

3.6 Clock generation optimization

This optimization is rather simple; the two crystal oscillators used in the system are replaced by more efficiently oscillators to save power. The low frequency continuous running crystal, is replaced with the low power crystal oscillator from Werner Thommen [22]. This oscillator uses 81nW, roughly one fifth of the assumed oscillator used in the Chipcon. A RC oscillator could be optimized to use even less energy, but these oscillators do not have the needed accuracy to be used as a long term reference clock in a SoC. The high

[7]The instructions must be moved according to Formula 2.13, otherwise the high frequency crystal oscillator is started due to to term $\frac{\mathrm{N}_{\mathrm{instr}}^{\mathrm{sensor}} \cdot \mathrm{CPI}}{f_{\mathrm{processor}}}$ in the formula

frequency crystal oscillator is replaced with [10], which uses 75.1µW when activated and 270nJ to startup (against the previous 1.96mW and 144nJ). This oscillator has a higher startup power compared with the previous oscillator used, but this is being compensated by the lower power consumption while it is active.

The power consumption of the HFclock in the model is composed out of two parts: the high frequency crystal oscillator and the clock net. In this optimization only the crystal is changed, the power consumed by the clock net ($\mathrm{DC} \cdot 32\mathrm{MHz} \cdot 149\mathrm{pJ}$) stays the same.

Using Formula 2.13 on page 22 the DC is determined to be 0.38%. Formula 2.12 leads to the start-up frequency 5.1Hz. Using Formula 2.11 yields to power usage for the high frequency crystal oscillator (which is conform Figure 3.1 when adjusted with the DC:DC conversion efficiency):

$$\begin{aligned} \mathrm{P_{HF}/DC{:}DC_{eff}} &= (5.1\mathrm{Hz} \cdot 270\mathrm{nJ} + 0.0038 \cdot (75.1\mu\mathrm{W} + 32\mathrm{MHz} \cdot 149\mathrm{pJ}))/0.85 \\ &= 23.4\mu\mathrm{W} \end{aligned}$$

3.7 Program memory optimization

The last optimization is to go to the ultimate low power usage for this device on 180nm. In all previous systems FLASH is used to store the instructions, the core does an access to the FLASH only if the L1 SRAM, situated between the FLASH and the core, does not have the requested instruction. From paragraph 1.4.1 on page 9 it is known that ROM requires less active energy and leakage compared to FLASH, so the FLASH used as L2 in the system is replaced. The ROM memory is used to store both the instructions for the application processor and the radio processor and therefore it has the combined size of the previous used L1 instruction memory from both processors.

Two small (128 words of 32-bit) caches are placed between the processors and the ROM (one for each processor) as a L1 instruction cache to minimize the amount of accesses going to the ROM, this so called filter cache is explained in 1.4.1. Due to the small size of the filter cache it has only a hit rate of 85% [12], so 15% of the instruction fetches are still going to the ROM, where in all previous systems 10% L2 usage was assumed for the ARM (recall that the Pearl does not use the FLASH during run time). The energy used in the bus per transaction, to access the ROM stays the same because it is assumed that the length of the bus stays constant, which is true if the ROM is placed in the same location on the SoC as the FLASH.

A simplistic power estimation can only be done by filling in Formula 2.3 and 2.4 on page 19, which in fact is precisely what the model is doing. From Figure 3.1 it is possible to see that the power of the processor is decreased. The radio power also drops because it is including the processing part for the radio and the instruction for this now also are fetched from ROM instead of FLASH. The total power of the optimized complete system 90.5µW.

3.8 Summary

In this chapter various optimizations are executed on top-level and low-level decisions. Top-level decisions changes are executed on the sort of processor used in the system, the radio protocol, the way of processing data (batch processing) and exchanging the FLASH for storing the instructions with ROM (which in fact is a specification change rather than an implementation change). The low-level decisions turned out to change the implementation of the ADC, radio and clock generation. All the optimizations resulted in a power reduction from 3.2mW to 90.5µW, as depicted in Figure 3.1 on page 36.

4 Discussion

4.1 Radio assumptions

Two assumption are considered for the radio power:

- Only looked at the power of the WSN, not the complete network including network coordinator
- Interference of other radios

Some protocols, like CSMA, require the network coordinator to be continuously active in receive mode, even if no WSN is transmitting data. This thesis uses only the power consumed by the WSN and not the complete network which includes the network coordinator. It might be the case that the network coordinator is not able to run solely on scavenged energy.

Other radios on the same frequency as the WSN, can disturb the transmission due to collisions. If this happens the data needs to be retransmitted. In this thesis it is assumed that collisions do not happen, so the extra power consumption induced by collisions is not taken into account.

A detailed study on the complete power consumption of the radio, including collisions and different protocols, is outside the scope of this thesis and can be found in [8]. However, the results in [8] indicate that the power can increase when more nodes are added to the network. How much the average power increases depends on the protocol used in the network. The power for the CSMA protocol, executed on the Chipcon 2430 SoC, is on average 6.5mW per node when 8 nodes are active in the network, the power increases to 7.5mW on average per node if 32 nodes are in the network. The unmodified GTS protocol has 8 private slots for different nodes, with 8 nodes in the network the average power per node is 8.2mW, when 32 nodes need to share the 8 private slots collisions occur increasing the average power per node to 13.8mW.

4.2 Processor state retention

Batch processing could also be used to power down the memories of the processor to save leakage. The state of the processor must in this case be saved to FLASH, saving the state to another SRAM is only an option if the retention SRAM has less leakage than the SRAM from where the data originates. With the sampling rate of 200Hz or the batch rate of 4Hz it is not possible to use FLASH to store the state of the processor, due to limited erase/write cycles, as already explained in paragraph 1.4.1. However, there are application where the frequency of processing is much lower. Think for example about a temperature sensor, this only needs to be measured every 10 seconds. Processing is done once per 15 minutes. With an FLASH endurance of one million erase/write cycles and writing only once every 15 minutes the state of the processor to the FLASH it is able to work for 28.5 years. The usage of FLASH in these cases is an interesting idea due to leakage of the memory saved, however, this technique is only feasible if the leakage saved is more than the energy needed to copy the data to the FLASH. Explorations of state saving can be done with the model, this is however not applied to the optimization discussed in the previous chapter because the frequency of the needed state saving is to high to make use of FLASH.

4.3 ADC at higher speeds

The power used by the sensor in the last optimization from the previous chapter is 126.5nW at a 200Hz sampling rate. Other applications, like electromyograph (EMG), require a kHz sampling rate. The power usage of a sensor acquiring samples at a rate of 1kHz is calculated by Formula 3.6:

$$\begin{aligned} \mathrm{P_{sensor}} &= 1000\mathrm{Hz} \cdot (500\mathrm{ns} \cdot 4\mu\mathrm{W} + 84\mathrm{pJ}) + 90\mathrm{nW} \\ &= 176.0\mathrm{nW} \end{aligned}$$

This increased power usage is not a problem, even multiple sensors could be used without affecting to total power usage considerately. If for instance, ten ADCs are acquiring samples at a rate of 1kHz, they will consume 1.76µW. The rest of the system consumes 90.5µW, the total power used is in this case still below the 100µW limit of energy savaging. Note that in this demonstration only the sensor power is changed, an increased sampling rate would normally also lead to an increased activity of the processor due to an increased work load. There is still 100.00µW − 92.26µW = 7.74µW left for processing, so even in the case of the multiple sensors there is still room to do some extra processing.

4.4 Power gating

Power gating on 180nm does not have a huge impact because the leakage is not the mayor factor in the total power usage. While the feature size shrinks the leakage will increase and the active energy will decrease. The leakage may become dominant in low DC systems made in smaller feature sizes. This will make power gating more important in the future.

4.5 ROM as an instruction memory

Using ROM in the complete optimized system from the previous chapter is not a practical solution because the device can not be reprogrammed. The created platform in Chapter 3 is also able to run other low duty cycled applications if another way of storing the program is used. Reprogramable non-volatile memory, like FLASH, is in this case the best choice to store the instructions for the processor. It has no leakage when powered off (during the sleep state) but uses more energy in active mode and has a higher leakage compared to ROM. If ROM cannot be used in the system the last optimization from the previous chapter should be ignored.

4.6 Change crystal earlier

The order of optimizing in the previous chapter is from the most to the least power consuming block. A careful inspection of Figure 3.1 shows that high frequency crystal oscillator uses the most energy after optimizing the processor and radio. This should be the logical choice to optimize next. This is actually done, but the result was that the system uses more energy after changing the oscillator. This was caused by the amount of startups of the oscillator. The new oscillator uses more energy to startup while the active power is less. This resulted into a power increase from 135.0µW to 148.5µW.

The optimization was undone and other optimizations are applied. After applying batch processing as an optimization, the amount of high frequency startups was reduced allowing another try to change the crystal. The switch of oscillator was in this case beneficial; a 48.3µW to 23.4µW power drop in the HF-clock, as seen in Figure 3.1.

4.7 Effective energy per bit

The expression "energy per bit" is a popular way to express radio efficiency. System designers might get mislead by this number because it does not take the protocol into account which has a big impact on energy per bit in real life situations. The formula to calculate the energy used by the protocol for transmitting a packet (without data) is shown in Formula 2.7 on page 21, while the energy used to transmit the payload is stated in Formula 2.8.

The Chipcon radio is used to demonstrate the effect of the CSMA protocol on the energy per bit. When Formula 2.7 is used with the power numbers of the Chipcon and the CSMA protocol parameters, it shows that the cost of transmitting a packet is 198.4µJ, while Formula 2.8 calculates that transmitting 10 bytes as payload uses 10.6µJ.

Normally the advertised Chipcon energy per bit (only taking the radio part into account) is $33.2\text{mW}/250\text{kbps} = 133\text{nJ/b}$, however, if the protocol is taken into account the energy per bit becomes $(198.4\mu\text{J}+10.6\mu\text{J})/(10\text{bytes} \cdot 8) = 2.6\mu\text{J/b}$. This huge difference is caused by the low amount of payload bytes, however when the maximum packet size of CSMA (88 bytes) is used, it becomes 414.6nJ/b, still more than tree times the advertised energy per bit.

5 Conclusions

Model creation
A layered model is created for system design exploration of WSN on power usage. The model is able to forecast power usage and is used to check if energy scavenging is feasible for a typical wireless physiological body sensor like an ECG.

The input of the model is the architecture of the system and the application being executed. The output is the average power consumption. A layered decomposition of the architecture is done to create functional blocks like:

- Processor
- Radio
- Power manager
- Sensors
- High frequency clock

The application is characterized with the next input parameters:

- Instructions for acquiring a sample
- Instructions for radio processing
- Instructions for application processing
- Memory usage
- Sample frequency
- Transmitted bytes per second

- Transmit interval
- Extra oscillator start ups
- Wireless protocol parameters

The architecture blocks are described in energy per event, leakages, timings and power usages. The model maps the application to the architecture for calculating the power. Bottlenecks on system level power consumption can be identified for the application. The optimal architecture is driven by the application and can be found with the help of the model in an iterative process where bottlenecks are tackled one after each other.

Validation of the model
Validation of the model can not be done straightforward because the Chipcon 2430 Soc [1] does not allow to measure the power consumption of the separate blocks as specified in the model. The measured power of the Chipcon is broken up in parts so that it can be assigned to the different components in Section 2.3.2. Two applications are run to prevent fitting of the model parameters to one application, the application parameters are described in Section 2.3.1.

The difference of the model compared to the measurements is 0.4% for the RAW ECG application.

The 2nd application is an ECG parameter extraction algorithm, the power bottleneck in this application is located in the processor and not in the radio as in the previous application. The output of the model for this application differs 1.4% when compared to the measurement. This shows that the hardware parameters are not fitted to the first application and that the model works correctly.

Complete optimized system
When a new optimized system, locally calculating the ECG parameters and sending the data wireless, is created from scratch it will consume 90.5µW; a reduction of 35 times in power compared to the original Chipcon implementation, see Figure 3.1. This shows that it is possible to let a WSN run on the 100µW power budged provided by energy scavenging. This has been realized with the next system where the several tasks are represented in groups:

- Processing:
  Optimized Pearl processor [3] for application processing
  ROM used for storing instructions, a small high Vt SRAM used as instruction filter cache
  High Vt SRAM for data storage
  HF crystal oscillator to generate the clock for the processor
  Processing in batches to decrease the startup power of the crystal

- Acquiring data:
  Ultra low power ADC which has it own clock generator
  Samples stored in local ADC memory to enable batch processing

- Radio:
  Ultra low power radio from Philips
  High frequency crystal oscillator (same used for processor) with PLL for the radio
  GTS protocol [8] executed on a
  Asynchronous RISC processor [6]

- Power management:
  Continuously running low frequency crystal oscillator[11]
  DC:DC conversion efficiency of 85%
  Power gating all components[1]

General purpose system
Application specific design of a system is the key to get to the power level where energy scavengers are able to power the system. Trade-offs must be made on programmability versus power. Take for instance the complete optimized system, it has a ROM to store the application data. Once ROM is programmed (during the fabrication) it cannot be altered, meaning that the device is fixed to its destined application. The use of a mask programmable ROM makes it possible to change the application during manufacturing without creating a complete new chip layout, therefore saving costs. However, devices with FLASH are preferable so that the application can be changed after fabrication. This is the key to make a general purpose system. FLASH consumes more power than ROM as seen in the last optimization steps of the previous chapter.

5.1 Summary

This study is a first attempt to get global insight if a system is possible to run solely on scavenged energy. The created model works correctly and using the model it is showed that an QRS extraction algorithm can run on scavenged energy More detailed studies are needed to substantiate these findings.

[1]The start-up power of hardware not put in the model for all components because these values are unknown, although the model is ready to have the values as an input

6 Future work

6.1 Peripherals

On of the first extensions to the model would be to make it easier to add peripherals to the model. Sensors in the current model would be a part of the peripherals. In the current model, the sensors are already able to model most characteristics of peripherals. In the current model, the sensors share a frequency they are activated on, which is a problem if multiple peripherals are used that should be activated on their own frequency. This problem can be fixed by giving every peripheral a private frequency. To enable easy exchange of peripherals, the amount of instructions per peripheral should be given as an input. In the current model, the total instructions to handle all the sensors are given as an input, which might change of an different peripheral (sensor) is selected. Every peripheral should be able to select a processor on which the instructions are executed, which in the current model is always the same processor.

6.2 DC:DC efficiency

DC:DC converters do not have a single efficiency, the efficiency depends on difference between the input and output voltage and on the current output, as illustrated in Figure 6.1. The efficiency in the low current consumption could be a problem for low duty cycled systems because these systems spend most of the time with a low current consumption. A power dependent DC:DC efficiency is not implemented in the model and could be an improvement of the total predicted power output of the model.

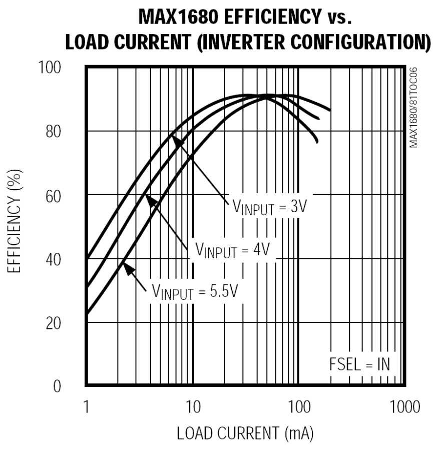


Figure 6.1: The efficiency of the MAX1680

Bibliography

[1] Texas Instruments [2007]. *A True System-on-Chip solution for 2.4 GHz IEEE 802.15.4 / ZigBee, Rev. 2.01 (SWRS036E)*, http://www.ti.com

[2] NXP [2007]. *Memory portfolio*, http://nww.nxp.com/it/cto/products/php/memories/memories_page.php

[3] Michael de Nil [2007]. *Ultra low power ASIP design for wireless sensor nodes*, Thesis Technical University Eindhoven

[4] Lennart Yseboodt [2007]. *System level design of 100 μW wireless sensor nodes for biomedical monitoring*, Thesis Technical University Eindhoven

[5] Handshake Solutions. *Asynchronous 8051*, http://www.handshakesolutions.com/products_services/HT-80C51

[6] Handshake Solutions. *Asynchronous ARM*, http://www.handshakesolutions.com/products_services/ARM996HS

[7] Michiel Elzakker [2006]. *Low power analogue to digital converter*, Thesis University Twente

[8] Alejandro Mendoza García [2007]. *Power management of wireless sensor networks in medical application*, Thesis Technical University Eindhoven

[9] EM MicroElectronic [2007]. *Low Power Crystal Oscillator 32.768 kHz*, http://www.emmicroelectronic.com

[10] Michiel Elzakker [2007]. *Crystal for ultra low power radio*, Philips Research

[11] Epson Toyocom [2007]. *Low Power Crystal Oscillator 32.768 kHz*, http://www.epsontoyocom.co.jp/english/

[12] Johnson Kin, Munish Gupta and William H. Mangione-Smith [1997]. *The Filter Cache: An Energy Efficient Memory Structure*, The Department of Electrical Engineering, UCLA, University of California

[13] Weiyu Tang, Rajesh Gupta and Alexandru Nicolau [2001]. *Power Savings in Embedded Processors through Decode Filter Cache*, Department of Information and Computer Science, UCLA, University of California

[14] Weiyu Tang, Rajesh Gupta and Alexandru Nicolau [2000]. *Design of a Predictive Filter Cache for Energy Savings in High Performance Processor Architectures*, Department of Information and Computer Science, UCLA, University of California

[15] J. Hennessy and D. Patterson[2003]. *Computer Architecture, a quantitative approach*, Morgan Kaufmann publishers

[16] Cheol Hong Kim, Sung Woo Chung and Chu Shik Jhon [2006]. *PP-cache: A partitioned power-aware instruction cache architecture*, Seoul National University and Korea University

[17] Lin Li, Ismail Kadayif, Yuh-Fang Tsai, N. Vijaykrishnan, Mahmut Kandemir, Mary Jane Irwin and Anand Sivasubramaniam [2003]. *Managing Leakage Energy in Cache Hierarchies*, Pennsylvania State University

[18] Stefanos Kaxiras, Zhigang Hu and Margaret Martonosi [2001]. *Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power*, Agere Systems and Princeton University

[19] Holst Centre. *Open Innovation by IMEC and TNO*, http://www.holstcentre.com/

[20] Dragan Maksimovic and Sandeep Dhar. *Switched-capacitor DC-DC converters for low-power on-chip applications*, University of Colorado

[21] Maxim [1997]. *Switched-Capacitor Voltage Converters*, http://maxim-ic.com

[22] Werner Thommen. *An improved low power crystal oscillator*, Microdul

A 8051 to ARM: instruction count

In this appendix, the architectural differences of an 8051 and ARM are addressed that motivate the factor six difference in instruction count for data algorithms like QRS processing in section 3.2.1 and a factor 4.5 for control algorithms.

The 8051 is based on an 8-bit Complex Instruction Set Computer (CISC) core with Harvard architecture and originally developed as a controller. With the 8 bit processor it is possible to calculate 16 or 32-bit arithmetic operations but this comes with a heavy burden. To illustrate this, the overhead of an addition (ADD) and multiplication (MULL) is detailed in Table A.1. The 8-bit addition operation from Table A.1 is explained in the accumulator architecture of the 8051 in part a of Figure A.1. One of the operands in the accumulator based architecture comes from the register (REG or memory) while the other comes from the accumulator (ACC). The accumulator needs to be filled with one operand before a two operand instruction can be executed. This is done by using a move instruction, as shown in the Table A.1 and Figure A.1 with a 1. The result of the addition is written back to the accumulator (2 in the table and figure), which is followed by copying the data from the accumulator with a move instruction (3) to a register (or memory). So the 8-bit addition operation takes only three instructions on the 8051 but the amount of instructions will repidly increase for larger operand widths. For example, Table A.1 illustrates that 16 and 32-bit addition operation take respectively 6 and 18 instructions to complete on the 8051. For multiplications, the amount of instructions depend even stronger upon data width as illustrated in Table A.1 as well.

By adding the clock cycles for all instruction listen in Table A.1, one arrives at the total number of clock cycles to perform the ADD and MULL operations, which is displayed in the second column of Table A.2, while in column four, the number of instructions for an ARM is depicted.

#	ADD 8 bit	ADD 16 bit	ADD 32 bit	MUL 8 bit	MUL 16 bit	MUL 32 bit
1	MOV A,R2	MOV A,R4	MOV A,@R0	MOV A,R1	MOV V0,R2	MOV V0,R2
2	ADD A,R1	ADD A,R2	ADD A,@R1	MOV B,A	MOV V0,R3	MOV V0,R3
3	MOV R1,A	MOV R2,A	MOV @R0,A	MOV A,R2	MOV A,R4	MOV A,R4
4		MOV A,R5	INC R0	MUL AB	MOV R2,A	MOV R2,A
5		ADDC A,R3	INC R1	MOV R1,A	MOV A,R5	MOV A,R5
6		MOV R3,A	MOV A,@R0		MOV R3,A	MOV R3,A
7			ADDC A,@R1		MOV A,R2	MOV A,R2
8			MOV @R0,A		MOV B,V0	MOV B,V0
9			INC R0		MUL AB	MUL AB
10			INC R1		XCH A,R2	XCH A,R2
11			MOV A,@R0		MOV R4,B	MOV R4,B
12			ADDC A,@R1		MOV B,V0	MOV B,V0
13			MOV @R0,A		MUL AB	MUL AB
14			INC R0		ADD A,R4	ADD A,R4
15			INC R1		MOV R4,A	MOV R4,A
16			MOV A,@R0		MOV B,V0	MOV B,V0
17			ADDC A,@R1		MOV A,R3	MOV A,R3
18			MOV @R0,A		MUL AB	MUL AB
19					ADD A,R4	ADD A,R4
20					MOV R3,A	MOV R3,A
...						...
...						...
107						MOV R2,A

Table A.1: Instruction count for an addition and multiplication on a 8051 for different data width

	8-bit 8051 Chipcon 2430		32-bit RISC ARM9 HS	
Operations	#instr	clock cycles	#instr	clock cycles
ADD 8-bit	3	4	1	1.5
ADD 16-bit	6	8	1	1.5
ADD 32-bit	18	40	1	1.5
MUL 8-bit	5	10	1	1.5
MUL 16-bit	20	48	1	1.5
MUL 32-bit	107	244	1	1.5

Table A.2: Summary of instruction count to execute an addition and multiplication on a 8051 and RISC

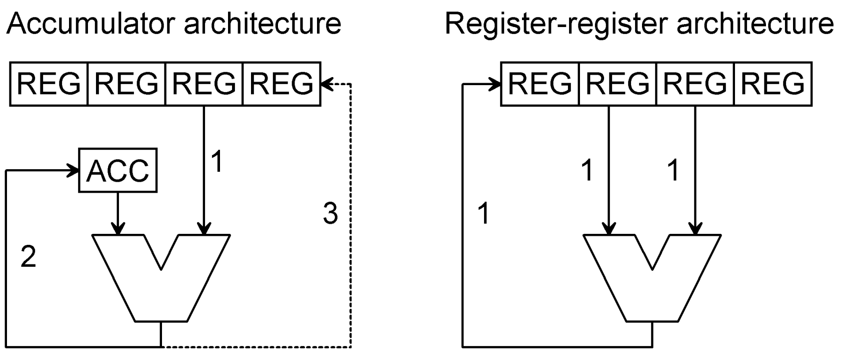


Figure A.1: Architecture of 8051 (a) and RISC (b), numbers indicate instructions executed to do an addition

The 32-bit register-register architecture of the Reduces Instruction Set Computer (RISC) makes it possible to execute every addition and multiplication (up to 32-bit) in only one instruction, eliminating the MOV, INC and other instructions needed by the accumulator architecture to execute the operation. Figure A.1 illustrates how this is accomplished in the RISC architecture. Table A.2 shows a six-fold reduction in instruction count for an ADD16. Since the QRS extraction algorithm is written for 16-bit wordsizes and contains many two 16-bit operand operations like ADD16, the reduction in amount of instructions for 16-bit data is assumed to be a factor six. Not every operation uses two operands (i.e. A=10) and therefore, will not have a six time difference in instructions However, these are compensated by the more expensive operations like the multiplication (Table A.2 shows a 20 times difference).

Control applications do not have a six time reduction in instruction count because it is a mix 8-bit and 16-bit operations, Table A.2 lists a factor 3 and 6 respectively for these operations. It is assumed that the ratio for 8-bit and 16-bit operations in the MAC processing is equal for both parts, this leads to the average instructions decrease for the ARM compared to the 8051 of: (3+6)/2=4.5.

B Cycles Per Instruction (CPI)

In this Appendix, the cycles per instruction are determined for a sample code executed on the 8051 processor embedded in the Chipcon 2430 SoC. This CPI will be used in Section 2.3.1 to estimate the amount of executed instructions from measurements of active times.

Operations written in a high-level program language, like C, will be compiled to instructions for the processor. The CISC architecture of the 8051 needs one, two or more machine cycles to execute an instruction depending on the instruction complexity and memory location of the operands, which can be seen in Table B.1. For example, 'MOV A,R' a single machine cycle whereas 'MOV @R,A' uses multiple machine cycles. The implementation of the processor will specify how many clock cycles are needed to execute a machine cycle. In the case of the Chipcon, it uses one clock cycle to execute one machine cycle. Table B.1 shows a selected summary of Chipcon's 8051 instruction set together with the number of clock cycles required to execute each instruction.

The average amount of clock cycles needed to execute a set of instructions will determine the Cycles Per instruction (CPI). To determine the CPI of the Chipcon, part of the ECG parameter extraction algorithm from [4] is compiled and the resulting instructions and lock (machine) cycles analyzed. The code is unrolled manually so that the real amount of executed instructions is counted. In this process the loop overhead is copied too, so that the control part of the program is also taken into account. Because the duration of an instruction can differ in amount of machine cycles, these had to be counted as different instructions. The result of counting the instructions can be seen in Table B.2 where the instructions are grouped together. The CPI is calculated as the weighted average of all the CPI's and in the case of the Chipcon the CPI for the instructions is 2.46.

In the CPI calculation of the Chipcon the prefetch of the instruction code is not taken into account and this needs to be added to the CPI. The 8051 has

Instruction	Description	Opcode	Bytes	Cycles
MOV A,R	Move register to accumulator	E8-EF	1	1
ADD A,R	Add register to accumulator	28-2F	1	1
MOV R,A	Move accumulator to register	F8-FF	1	2
ADDC A,R	Add register to accumulator with carry	38-3F	1	1
MOV A,@R	Move indirect ram to accumulator	E6-E7	1	2
ADD A,@R	Add indirect ram to accumulator	26-27	1	2
MOV @R,A	Move accumulator to indirect ram	F6-F7	1	3
INC R	Increment register	08-0F	1	2
ADDC A,@R	Add indirect ram to accumulator with carry	36-37	1	2
MULL A,B	Multiply A B	A4	1	5
XRL direct,#data	Exclusive OR immediate data to direct byte	63	3	4
DIV A,B	Divide A B	84	1	5
MOVX @R,A	Move accumulator to external RAM	F2-F3	1	4-11
PUSH	Push direct byte to stack	C0	2	4
LJMPaddr16	Long jump	02	3	4
ANL A, R	AND register to accumulator	58-5F	1	1
ANL direct, A	AND accumulator to direct byte	52	2	3
ORL A, @R	OR indirect ram to accumulator	46-47	1	2
ORL direct,#data	OR immediate data to direct byte	43	3	4
XRL A, #data	Exclusive OR immediate to accumulator	64	2	2
XRL direct, A	Exclusive OR accumulator to direct byte	62	2	3

Table B.1: Summary of Chipcon's 8051 instruction set with HEX opcode, instruction bytes and machine/clock cycles

Instruction	Total used	8051 CPI distribution	CPI
MOV	39546	19605 of 1 CPI, 19941 of $\geq$ 2 CPI	1.84
ADD	7312	4604 of 1 CPI, 2708 of 2 CPI	1.37
MOVX	5114	variety of cycles, ranging from 3 to 11	8
MULL	4956	5 CPI	5
ADDC	4007	2402 of 1 CPI,1605 of 2 CPI	1.4
XCH	1652	2 CPI	2
SUBB	1500	800 of 1 CPI, 700 of 2 CPI	1.47
Others	7064	1971 of 1, 2888 of 2 1102 of 3 and 1103 of 4	2.19
Total	71151	weighted average	2.46

Table B.2: CPI distribution of Chipcon 8051 instructions for part of the ECG extraction algorithm

a 4 byte L1 buffer[1] to store instruction code and filling this buffer takes one clock cycle. The unrolled program uses 71151 instructions and 103838 code bytes. This information also tells us the amount of code bytes on average per instruction: 103838/71151 = 1.46, this number is used to determine the activity of the L1 instruction memory which is used to calculate the energy used for fetching the instruction code. To the already determined CPI of 2.46 we need to add the average cycles per instruction for prefetching to determine the total CPI for the Chipcon:

$$\text{CPI} = 2.46 + \frac{103838\text{bytes}/4}{71151\text{instr}} = 2.82$$

In principle, the RISC architecture makes a CPI of one possible, but in practice, the pipelined architecture causes an increase in the average CPI due to branches and stalls. The CPI of an ARM9 is 1.5[2], as can be seen in Table A.2.

Another 8051 is examined to illustrate that the CPI is implementation depended. This is done by taking a Nordic 8051 and repeating the above described algorithm mapping exercise. This 8051 uses 4 clock cycle to process one instruction cycle. Most instructions use the same amount of machine cycles when the Nordic is compared with the Chipcon. Due to the Nordic using 4 clock cycles for one machine cycle, it is expected that the Nordic has around a 4 times higher CPI than the Chipcon. The calculation revealed that the Nordic has a CPI of 8.22, indeed a rough 4 time difference with the Chipcon.

[1]Chipcon data sheets [1] states that four bytes are preloaded in an instruction fetch
[2]Taken from http://www.arm.com/support/faqdev/4160.html