Power Optimization under Performance Constraints for Periodic Tasks with Soft Deadlines

Matias Pitkänen

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 29.7.2019

Supervisor

Senior University Lecturer Vesa Hirvisalo

Advisor

M.Sc. Jani Marjamaa

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Matias Pitkänen

Title Power Optimization under Performance Constraints for Periodic Tasks with Soft Deadlines

Degree programme Computer, Communication and Information Sciences

Major Computer Science **Code of major** SCI3042

Supervisor Senior University Lecturer Vesa Hirvisalo

Advisor M.Sc. Jani Marjamaa

Date 29.7.2019 **Number of pages** 57 **Language** English

Abstract

This thesis aims to find the best power manager for a system where a single device is connected to multiple external devices through a shared blocking communication line. We consider a scenario where the external devices send periodic messages. The device we examine can transition between different run and sleep modes whereas the external devices stay in a predetermined run mode when active and in a predetermined sleep mode during inactive sections. We aim to minimize the power consumption of the entire system by utilizing the different power modes. We construct different power managers that control the power mode transitions.

We perform measurements on an example microcontroller to acquire the necessary characteristics of the device. We measure the current consumption and the communication bus speed in different run and sleep modes along with the transition times between different power modes. We use these values to analyze the different power managers we propose.

We present six different power managers. Each of the power managers use a different technique to decide which power modes to use. We present the different power managers and give algorithms to calculate the power mode schedules. Then we compare the power managers using example systems and using the device characteristic from the microcontroller we measure. We find that the power savings for the microcontroller we is use is modest, about 10 percent at most. Only the most efficient power manager displays this power saving. This power manager is also the most complex. The simpler power managers display a lower power saving, in the range of single percent savings. The default power manager which the comparisons are done against is a system which uses one run mode and one sleep mode and always enters a sleep mode when inactive.

Keywords Embedded, power management, low power, dynamic power management, dynamic frequency scaling

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Matias Pitkänen

Työn nimi Jaksollisten tehtävien suorituskykyrajoitteinen virtaoptimointi pehmeillä aikarajoilla

Koulutusohjelma Computer, Communication and Information Sciences

Pääaine Tietotekniikka **Pääaineen koodi** SCI3042

Työn valvoja Vanhempi yliopistonlehtori Vesa Hirvisalo

Työn ohjaaja DI Jani Marjamaa

Päivämäärä 29.7.2019 **Sivumäärä** 57 **Kieli** Englanti

Tiivistelmä

Tämän diplomityön tavoite on löytää paras virranhallintajärjestelmä systeemille, missä yksi laite on yhdistetty moneen ulkopuoliseen laitteeseen yhteisen jaetun kommunikaatioväylän kautta. Me tarkastelemme systeemiä, missä ulkoiset laitteet lähettävät jaksollisia viestejä. Tarkastelemamme laite voi siirtyä eri prosessointitilojen välillä, kun taas ulkopuoliset laitteet pysyvät tietyssä ennaltamäärätyssä prosessoritilassa ollessaan aktiivisia ja ovat muuten tilassa, jossa virrankulutus on nolla. Meidän tavoite on minimoida virrankulutus koko systeemissä käyttämällä eri prosessoritiloja. Me luomme erilaisia virranhallintajärjestelmiä, jotka hallitsevat siirtymät prosessoritilojen välillä.

Suoritamme mittauksia esimerkkimikrokontrollerille, jotta saamme tarvittavat laitteen ominaisuudet. Me mittaamme virrankulutusta prosessorin eri suoritusmoodeissa ja mittaamme siirtymäajat eri suoritusmoodien välillä. Käytämme näitä mitattuja arvoja meidän esittämien virranhallintajärjestelmien analysointiin.

Me esitämme kuusi erilaista virranhallintajärjestelmää. Ne käyttävät eri tekniikoita päättämään mitä suoritusmoodia käytetään milloinkin. Me esitämme jokaisen virranhallintajärjestelmän ja annamme jokaiselle algorithmin, jolla voi laskea suoritusmoodiaikataulun. Me vertaamme virranhallintajärjestelmiä käyttämällä erilaisia esimerkkisysteemejä, joissa käytämme mikrokontrollerista mittaamiamme arvoja. Me huomaamme, että virransäästö käyttämällämme mikrokontrollerille on vaatimaton, noin 10 prosenttia maksimissaan. Vain kaikkein tehokkain virranhallintajärjestelmä saa tämän säästön. Tämä järjestelmä on myös kaikkein monimutkaisin. Yksinkertaisemmat virranhallintajärjestelmä säästävät vähemmän virtaa, suunnilleen muutaman prosentin luokkaa. Nämä virrankulukset ovat verrattuna virranhallintajärjestelmään, joka käyttää samaa suoritusmoodia aina ollessaan aktiivinen ja toista suoritusmoodia aina kun ei ole mitään prosessoitavaa.

Avainsanat Sulautettu, virranhallinta, vähävirtainen, dynaaminen virranhallinta, dynaaminen taajuusskaalaus

Preface

First I want to thank my thesis supervisor Professor Vesa Hirvisalo for providing support and guidance for the creation of this thesis. His advice was invaluable when it comes to the direction of the thesis and where it eventually ended up.

I want to thank Panu Kilponen and Jani Marjamaa and other Vaisala personnel who gave constructive feedback on the content of the thesis. I would like to thank Vaisala for providing me with the opportunity to do this thesis in their company and providing funding for this thesis.

I would also like to thank my family for supporting me throughout the entire process of writing this thesis.

Otaniemi, 29.7.2019

Matias Pitkänen

Contents

Abbreviations

ADC	Analog-to-digital converter
ARM	Advanced RISC Machine
CPU	Central processing unit
DAC	Digital-to-analog converter
DFS	Dynamic frequency scaling
DPM	Dynamic power management
DVFS	Dynamic voltage and frequency scaling
DVS	Dynamic voltage scaling
FIFO	First in, first out
GPIO	General-purpose input/output
IC	Integrated circuit
IDE	Integrated development environment
LLS	Low leakage stop
LLWU	Low-leakage wake-up unit
MCU	Microcontroller unit
RAM	Random-access memory
OPP	Operating performance point
SRAM	Static random-access memory
I2C	Inter-Integrated Circuit
SDA	Serial Data Line
SCL	Serial Clock Line
TSI	Touch sense input
USB	Universal serial bus
VLLSx	Very low leakage stop
VLPR	Very low power run
VLPW	Very low power wait

1 Introduction

Consider a device that is connected to multiple external devices through a blocking communication bus. For example this could be a gateway connected to multiple measuring sensors through a common wired connection. These sensors send data periodically to the gateway and the gateway needs to receive and store this data. We assume the device has no function other than receiving messages from other devices connected to it. Keeping the device on the highest activity state would often be wasteful because incoming messages might be infrequent and thus the device would end up unnecessarily consuming a lot of power during periods when no messages are being sent. It is beneficial for the receiving device to utilize sleep modes and other low power modes to reduce the power consumption when less processing is required. However, it is not trivial to know which processor mode is the most efficient during a transmission or during an idle period between two transmissions.

The available power modes, their power consumptions, and the transition times between power modes vary greatly between different devices so we use an example device to measure realistic values for the given problem. We use a Teensy 3.2 microcontroller development board with an ARM Cortex-M4 microcontroller in the measurements. For the communication between different devices we use an I2C communication bus. This thesis only focuses on the device used in the measurements but most methods used in this thesis are also applicable for other devices.

First we measure the power consumption in different modes along with the bus speed and wake-up times of the device. Using these values we calculate all the necessary characteristics for the embedded device. We find that the measurement align closely but not exactly with the values in the data sheet of the measured device. Then we explain the power management model in terms of the relevant parameters and how they affect the system. We also establish the device topology we use. Then we propose different power managers both in terms of how the device should wake-up and enter sleep modes along with which processor modes the device should be using. We present algorithms for calculating the power schedules for each of the proposed power managers. With these algorithms we can calculate the power management schedules and power consumption for a given transmission system. Using the measurements and the power managers we perform simulations for different example systems. We find that the power savings vary from 0% to about 10% compared to a simple power manager where a single processor mode is used and where the processor always enters a sleep mode when idle and only wakes up when a message is being transmitted.

1.1 Problem Statement and Contribution

Embedded devices often are working in conditions where available powering options are limited so it is important to conserve as much power as possible. In this thesis we try to find efficient power managers for a given device. We investigate a system where the device is connected to a number of external devices. These external devices transmit periodic messages that we have no control over.

This thesis aims to find the most power efficient power managers for the system as a whole by controlling the receiving device. We do this by using different processor modes and sleep modes so that all of the devices are active as little as possible while at the same time trying to receive the messages at efficient speeds. The faster transmission speeds are not always also more efficient.

The main contributions of this thesis are the measurements of the embedded device and the discovery and analysis of the different power managers and the algorithms that generate the power schedules for the device. We create a physical measuring setup and use this setup to measure all the necessary characteristics of the device. We also formulate different power managers and give algorithms for calculating the power schedule for each of them. We briefly evaluate the efficiency of the different power managers with the measured device using simulations.

1.2 Structure of the Thesis

The relevant background is described in section 2. We introduce integrated circuits and microcontrollers with their power management techniques in general. In section 3 we present the measurement setup along with the measurement results. We describe the measurement setup in terms of both hardware and software. In the measurement results we present the current consumption in different modes along with the transition times between different modes and the communication bus speed. In section 4 we explain the power management model we use. Then we explain the different types of power managers we use. We compare the characteristics of the different power managers. Finally we present algorithms for calculating the power schedules for the different power managers. In section 5 we perform simulations for example message systems with the power managers we present. We compare the results from the simulation and analyze the effectiveness of each of the proposed power managers with each of the example message systems. In section 6 we assess how well the proposed methods solve the problem we state. Finally in the last chapter we conclude the work done in this thesis.

2 Background

In this chapter we cover some basic background necessary for this thesis. We describe the basic concepts of integrated circuits, microcontrollers and where energy and power is used in microcontrollers. We explain how power management is done in microcontrollers in general and what techniques there are to lower the energy and power consumption of a microcontroller.

2.1 Energy and Power

Energy is a quantitative property that expresses the amount of work a system is able to do to another system. The common symbol for energy is E and the SI unit of energy is joule (J). Power is the rate of which the energy being transferred. We denote power as P. The SI unit of power is watt (W). [19]

First we define average power as the energy transferred in a given interval:

$$P_{avg} = \frac{\Delta E}{\Delta t} \tag{1}$$

Then we define instantaneous power as the limit of the average power for when the time intervals approach zero [19]:

$$P = \lim_{t \to 0} \frac{\Delta E}{\Delta t} = \frac{dE}{dt} \tag{2}$$

We cannot measure the energy or power consumption directly from an electrical circuit. However, we can measure the current and voltage of the circuit. Instantaneous power can be described as a function of voltage and current using Ohm's law and Joule's first law [20]. Ohm's law states as follows:

$$V = R \cdot I \tag{3}$$

Where R is electrical resistance. Resistance is the measure of how difficult it is to pass current through an electrical component. The SI unit is resistance is the ohm (Ω). V is voltage which is the electric potential difference of two points. The SI unit of voltage is the volt (V). I is electric current which is defined to be the rate at which the electric charge flows [20]. The SI unit of electric current is the ampere (A). For brevity when we talk about current we are talking about electric current.

Joule's first law gives us the relation between power, current and resistance in electrical circuits:

$$P = I^2 \cdot R \tag{4}$$

By substitution we find that the instantaneous power for electrical circuits is:

$$P = V \cdot I = \frac{V^2}{R} \tag{5}$$

The energy consumed by a system over time interval T:

$$E = \int_0^T V \cdot I(t)dt \tag{6}$$

We can calculate the average power consumption with:

$$P_{avg} = \frac{E}{T} = \frac{1}{T}\int_0^T V \cdot I(t)dt \tag{7}$$

2.2 Integrated Circuits and their Power Consumption

Integrated circuits (IC) are electric circuits embedded into a single chip of semiconductive material. This allows the circuits to be magnitudes smaller and efficient compared to circuits made out of discrete components. Integrated circuits consist of a large number of tiny transistors integrated into a small chip.

The power consumption of an integrated circuit can be represented as the sum of the static power loss and dynamic power loss [32]:

$$P = P_{static} + P_{dynamic} \tag{8}$$

Static power loss is caused by leakage currents of the components in the integrated circuit like wires and transistors. Static power loss can be calculated with equation 9, where V_{dd} is the supply voltage and I_{sc} is the current flowing through the integrated circuit. The static power loss is always present regardless whether the transistors are switching. Static power loss mainly depends on the size of the IC. It can be reduced by disconnecting parts of the integrated circuit from the supply voltage or by reducing the supply voltage of the entire IC.

$$P_{static} = V_{dd} \cdot I_{sc} \tag{9}$$

Dynamic power loss can be calculated with equation 10, where β is the activity factor, C is the capacitance of the circuit, V_{dd} is the supply voltage, and f is the operating frequency of the processor. Dynamic power loss is mainly caused by the transistors switching states where the load capacitance of the IC is charged and discharged. So the dynamic power loss is proportional to the switching activity of the transistors. Dynamic power loss can be reduced by making parts of the integrated circuit nonactive or by reducing the supply voltage and/or operating frequency.

$$P_{dynamic} = \beta \cdot C \cdot V_{dd}^2 \cdot f \tag{10}$$

When attempting to reduce the dynamic power loss, it is common to attempt to reduce the power consumption by reducing the supply voltage, since the supply voltage impacts the power consumption quadratically. However, the voltage cannot be chosen arbitrarily. Every supply voltage has some maximum processor frequency it can support and in general the higher the wanted processor frequency the higher the supply voltage needs to be. [16]

2.3 Microcontrollers and Embedded Systems

Microprocessors are computer processors that implement the functionality of a central processing unit (CPU) on a single integrated circuit. Microprocessors require many external components including memory and peripheral interface chips in order to create a working system. As a result a total working system can become expensive. Microcontrollers (MCU) are a solution to this problem. A microcontroller typically contains a central processing unit, memory, and peripheral devices all on the same integrated circuit. There is memory for storing the program code, which is often flash memory, as well as RAM memory for the applications run on the microcontroller. Peripheral devices are used for getting information in and out of the microcontroller. Common peripherals found in microcontrollers are analog-to-digital and digital-to-analog converters (ADC) and (DAC), communication transceivers like I2C and USB, different timers and human-machine interfaces like GPIO and TSI [31]. These peripherals are accessible through the contact pins of the MCU.

Figure 1: A PIC 18F8720 microcontroller. (Retrieved from Wikimedia Commons [10])

Having all the components on the same chip allows them to be much cheaper to manufacture as opposed to creating all the components separately.

Usually microcontrollers are much more limited in terms of processing resources compared to multi-purpose microprocessors. Typically they operate at lower operating frequencies than traditional CPUs and have more restricted functionalities. This in turn means that they can have a very low power consumption, in some cases as low as microwatts which make them suitable for battery powered applications. They often also have the ability to receive an event in the form of an interrupt even when in a sleeping state.

Microcontrollers are generally designed for embedded applications. Typical uses for embedded systems are consumer electronics and household appliances like cameras

or washing machines. Embedded systems are also used in more complex systems like vehicles and automated factories.

2.4 Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) is a technique which aims to reduce the dynamic power consumption of a microprocessor by altering the processor frequency and supply voltage dynamically. Often the hardware has a discrete number of predefined operating performance points (OPP) which are voltage-frequency pairs in which the system can run [16]. This means the dynamic voltage and frequency scaling is done by stepping between these modes. What kind of OPPs are available depend heavily on the processor used.

If the frequency and voltage in a system are independent DVFS can be split into dynamic frequency scaling (DFS) and dynamic voltage scaling (DVS). In DVS the supply voltage is changed without affecting the clock frequency. A decrease in the supply voltage implies both lower power consumption and better processing efficiency.

In DFS the clock frequency is lowered without changing the supply voltage. Lowering the clock frequency has a negative effect on the execution times and latency as all the components of the integrated circuit function at a lower frequency. As can be seen from function 10 the power consumed is proportional to the clock frequency and because the processing speed is proportional to the operating frequency it means that the frequency change by itself has no impact on the efficiency of the CPU. This means that in theory it does not matter which frequency is used to calculate a time insensitive task. However, in practice this is often not the case because there are other factors that affect the processing speed and power consumption of the IC.

2.5 Dynamic Power Management

Dynamic power management (DPM) techniques aim to reduce the power consumption of the CPU by switching between different power modes. This could for example be using a sleep mode when the CPU has no tasks and using a run mode when there are tasks to be run. Typically CPUs have several different power modes. Often there are multiple different runs modes depending on the support of DVFS. Having many different sleep modes is also common; they can for example vary based on the depth of the sleep mode.

In DPM the transitions between different modes are also taken into account. Figure 2 represents the power state machine of a microprocessor with three different power modes. The figure shows the power consumption of each power mode and transition times between the different modes. There are usually large potential reductions in energy savings due to the fact that the sleep mode power consumption is usually a fraction of the power consumption of an active or an idle state. However, shutting down and restarting the processor has a cost associated with it which makes it hard to reach the full potential of the power savings of sleep modes. The large

power savings with dynamic power management are possible because both static and dynamic power loss can be decreased.

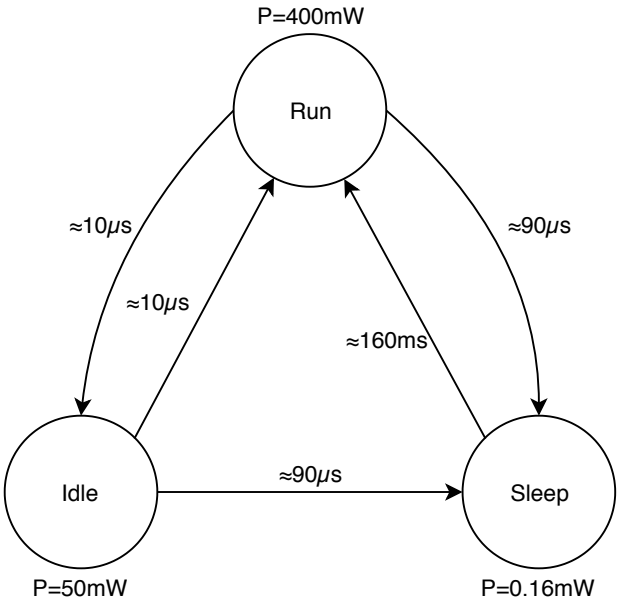

Figure 2: An example power state machine for a StrongARM SA-1100 processor, adapted from Benini [15].

2.6 Power Gating

Power gating is a technique where parts of the integrated circuits are shut off when they are not in use. This is done by cutting off the current from blocks of the IC. This reduces both dynamic and static power loss for that block. The downside is that additional functionality is needed on the hardware in order to support power gating. Also when power gating is used the block that was switched off needs its state stored. This is called state retention. When the block is turned on again the state of the block needs to be restored before it can continue operation. Because of this the time delays also increase when power gating is used.

2.7 Clock Generator and Clock Gating

Practically all microcontrollers have a clock generator in the integrated circuit. The clock generator produces a clock signal which is used to synchronize operations in a circuit. The clock signal oscillates between a high and a low state which coordinates the actions of different components in the chip. The clock generator dictates at which speeds the different components are running so by controlling the clock generator we can affect for example the core frequency of the CPU or control the speed of the memory.

The clock signal is transmitted to all the different components in the chip that need it. The clock pulses in the different components consume a significant portion of power even if the component is idling.

Clock gating is a technique where parts of chip are cut off from the clock signals. This reduces the power consumption of that component. However, in order for clock gating to be possible some additional logic is needed in the chips that enable it. Clock gating is only possible if the component is completely idle.

2.8 Workload Model

Typically in real-time systems, the computational workload is characterized by set of n periodic tasks $\{t_0, t_1, ..., t_n\}$ [14]. Each task is activated cyclically and each task generates a sequence of instances $t_{i,0}, t_{i,1}, ..., t_{i,m}$ we refer to as jobs. The jobs in a periodic task are separated by a period T_i so we can calculate the starting time for each job $t_{i,k}$ with:

$$s_{i,k} = O_i + k \cdot T_i \tag{11}$$

where O_i is the task offset.

Each task is t_i is characterized by an execution time $C_i(s)$ which is a function of the processor speed. A large portion of the related literature considers the jobs as fully scalable i.e. $C_i(s) = C_i/s$ although this is not always the case because of operations that are not affected by the clock speed like I/O peripherals or memory operations [14]. In cases like this a portion of the job is fully scalable and the other portion is fixed. This means its behavior can be modeled with Amdahl's law. Amdahl's law is usually used for parallel computing to compute the possible speedup when using multiple processors. However, it can apply to any job where only a part of it benefits from the increase in resources.

Amdahl's law is formulated as follows [11]:

$$Speedup = \frac{1}{(1-p) + \frac{p}{s}} \tag{12}$$

Where $Speedup$ is the speedup in the execution of the whole job, p is the proportion of the job which benefits from increased resources and s is the speedup of the part which benefits from the increased resources.

Most algorithms in research assume that the tasks are fully preemptive. This means the tasks can be suspended at arbitrary points in favor of a different task. This simplifies the schedulability but introduces an overhead during job processing. The overhead is due to the cost related to context switching between jobs. Nonpreemptive tasks always run to completion after they have started. This means the schedule has negligible runtime overhead but it can introduce delays for high priority tasks which hurts the effectiveness of the schedule. [14]

2.9 Power Management in Microcontrollers

Microcontrollers have many different options available for power saving, they provide system features which allow for example various run and sleep modes, smart peripherals which can remain operational during sleep modes, and clock system controls which allow clock signals to be turned off for inactive parts of the MCU [30]. Microcontrollers typically have many different sleep modes. The deeper the sleep mode is the fewer CPU and peripheral functionalities the microcontroller has. Generally deeper sleep modes have lower power consumption. However, lower power consumption also generally means a longer wake-up time.

A common approach to manage power in microcontrollers is to run the task quickly and then go to sleep as much as possible [30]. This is due to the static power consumption which is present regardless of the core frequency. If the static power consumption is high enough it is more efficient to process the task quickly and enter sleep quicker compared to running the task at a lower core frequency. Also processing tasks allows decent power savings while still allowing the processor to react to high processing demand if the tasks are not known in advance. Another well known method is timeout algorithms where if the device is idle for longer than some time t a sleep mode is entered [22]. This is based on the fact that if the device is idle for at least time t it is very likely that the device is idle for a long time.

Another algorithm used is the L-shape algorithm [28]. The algorithm is based on the assumption that a short active period is more often followed by a longer idle period. In the algorithm if the active period is short enough the device should enter a sleep mode. Both this and the timeout algorithms also work for systems with sporadic tasks meaning tasks that arrive randomly within some interval.

2.10 Interrupts

Interrupts are signals to the processor that some event needs attention. An interrupt could for example be caused by a button being pressed. The processor responds to this by suspending the current executing and run a function called the interrupt handler. There are two types of interrupts: software interrupts and hardware interrupts. With software interrupts the processor itself creates the interrupt for example during a divide-by-zero exception. Hardware interrupts are caused by external devices that want to communicate with the processor for example a keystroke on a keyboard could cause a hardware interrupt.

The interrupt is usually level-triggered or edge-triggered. Level-triggered means the interrupt is signaled by maintaining the interrupt line at a high or low voltage. Similarly edge-triggered interrupt is signaled by the rising or falling edge of the voltage.

2.11 Related Work

A lot of research has been done related to power management of real-time tasks. DVFS and DPM are widely used techniques in real-time systems for reducing energy

consumption. A common objective is deriving processor modes and speeds that guarantee the timing constraints of a system while minimizing the energy consumption. Equation 8 is commonly used in literature to represent the power consumption but other functions have been proposed to characterize the power consumption for example by Martin and Sieworek [24]. [14]

In this thesis we assume that the workloads are known in advance and that the work loads are periodic. In paper by Srivastava [28] they proposed algorithms for predicting workloads where then they are not known in advance. Aydin [12] gives a generalized energy management framework for periodic real time tasks. Luo [23] gives a power scheduling algorithm for systems with periodic and aperiodic tasks. Aydin [13] gives an algorithm for determining the optimal processor speeds for periodic real-time tasks with different power characteristics.

We do not allow delaying of messages in order to save power. Lee [18] proposed two algorithms for procrastinating task executions to compact idle intervals.

3 Measuring Setup

In this chapter first we introduce the measurement setup from the hardware and software side. Then we present the values measured and these values are analyzed.

The purpose of the measurements is to find the characteristics of an example microcontroller and use these characteristics to test the power schedules created by the different power managers and see how they behave. After we have done the actual measurements it is easier to know which characteristics matter the most and which have little impact.

3.1 Hardware

The test setup for measuring the power consumption of the embedded device is a Teensy 3.2 serving as the slave node for receiving messages. Teensy boards are USB-based microcontroller development systems [8]. Teensy 3.2 has a MK20DX256VLH7 microcontroller which is a 32-bit ARM Cortex-M4 MCU. It has a 72 MHz processor with 256 kB flash memory and 64 kB SRAM [3]. We use a Teensy LC as the master node sending messages to the slave node. In this experiment we use I2C as the protocol for message transfer. The Teensy boards have an I2C connection with the required SDA and SCL lines. The SDA and SCL lines are connected to +3.3 V with 4.7 kΩ pull-up resistors as recommended in the Teensy documentation [9]. The pins used for SDA and SCL in this test setup (Teensy 3.2 pins 18 and 19) are not configured for LLWU wake-up sources, therefore a different wake-up pin needs to be used when in low leakage modes (LLS, VLLS1, VLLS2, and VLLS3) [4, p. 208], [7]. LLWU or low-leakage wakeup unit is a module in the processor that allows external pin sources to wake the processor from low leakage modes. Low leakage modes are different deep sleep modes. In low leakage stop (LLS) mode most peripherals are in state retention mode and after an interrupt normal run mode is resumed after the LLWU interrupt routine. In very low leakage stop (VLLSx) modes most peripherals are disabled, the different VLLS modes vary by which portion of the SRAM is powered on [4, p. 182]. After an interrupt in a VLLSx mode the processor goes through a normal reset flow. In this test setup we use the Teensy 3.2 pin 7 as the wake-up pin.

Both Teensy boards have a serial connection and powering through a USB cable. In the Teensy 3.2 board we cut the solder pad in the power input line and placed a multimeter in between the pads so that the current used by the board can be measured [7]. The hardware schematic is illustrated in figure 3 and the actual test setup in figure 4.

All measurements were done at room temperature ≈ 22 °C.

All measurements are made in amperes which can easily be converted into watts with the electric power formula:

$$P = V \cdot I.$$

Where P is power, V is voltage and I is electric current.

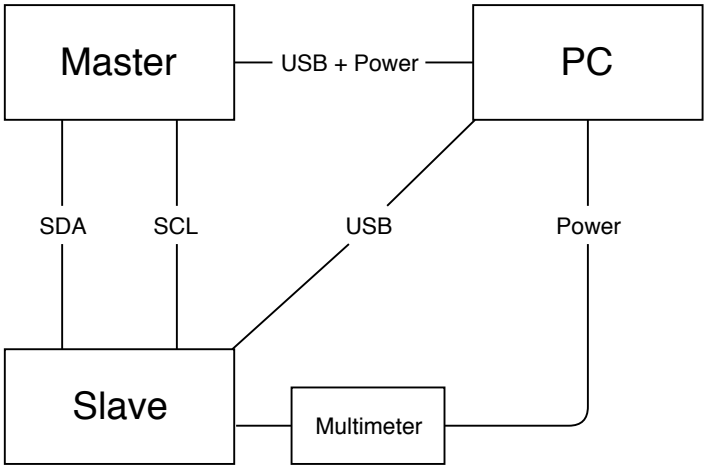

Figure 3: The hardware architecture used in the measurements.

Figure 4: The test setup used in the measurements.

Since the voltage remains roughly the same across all measurements (measured around 5.05V) the current alone is sufficient to measure the differences in power consumption in different modes. This is because only the relative differences in power consumption are of interest. Of course if the power consumption is ever needed the current value can be simply multiplied with the constant voltage. The voltage we measured is considerably higher than the 3.3V stated in the data sheet under typical conditions [3, p. 8]. This is because the USB provides a supply current of 5V which is

lowered to 3.3V with a linear regulator and the voltage measurement is done before the regulator.

3.2 Software

We created the software for the test boards using the open-source Arduino Software IDE. Arduino handles everything from compiling to uploading the binary to the microcontroller. Within Arduino IDE there is an option to define at which core frequency the software for the Teensy board will be compiled with. We used this option to measure the power consumption with different core frequencies.

In all the measurements we use a Teensy LC as the I2C master and the sender of the transmission. We use a Teensy 3.2 as the receiver in all the measurements and whenever we mention a core frequency it is always referring to this device. We do not do any measurements for the Teensy LC because the power consumption of that device is not relevant for the system we propose.

We use the "i2_t3" library for the I2C communication between the test boards. The library allows configuring the pins for the I2C communication, the I2C bus frequency, the I2C operation mode and the addresses of the devices in the bus. [1]

We use the "Snooze" library for testing different sleep modes. In the Snooze library REDUCED_CPU_BLOCK, sleep, deep sleep and hibernate are the processor modes VLPR, VLPW, LLS and LLS with USB voltage regulator turned off respectively [5]. In very low power run (VLPR) the on-chip voltage regulator is set to a low power state and the maximum core and peripheral clock frequencies are reduced. Very low power wait (VLPW) is the same as VLPR except the processor core is in sleep mode [4, p. 182]. In deep sleep and hibernate modes the low leakage mode can be manually specified (one of LLS, VLLS1, VLLS2, and VLLS3). We measure the relevant parameters for all these modes.

3.3 Measurements

We use amperes instead of watts in the power consumption comparisons since we assume that the supply voltage remains constant. We also see this in the measurements.

The core frequency has no impact on power consumption on these modes because the platform and peripheral clocks stop in VLPS, LLS, and VLLSx modes. VLPW and VLPR modes limit the system and bus clocks to 4MHz. [4, p. 186]

The values in table 2 are current consumption under typical conditions. The measurements in the data sheet for VLPR use 4MHz clock speed [3, p. 16]. Using lower core frequency decreases the supply current drawn so we can assume that if a lower core frequency was used the current drawn would also be lower in the data sheet.

The currents listed in the data sheet are significantly lower than the measurements we get in table 1. The Teensy board has additional components in addition to the actual MCU and this might be the reason why the supply current drawn is higher on the Teensy board than what the data sheet states.

Table 1: Current consumption measurements of different low power modes.

Power mode	Current consumption
VLPR (2MHz)	1.5 mA
VLPW (2MHz)	1.3 mA
VLPS	1.3 mA
LLS	0.25 mA
VLLS1	0.25 mA
VLLS2	0.25 mA
VLLS3	0.25 mA
LLS without USB voltage regulator	0.05 mA
VLLS1 without USB voltage regulator	0.05 mA
VLLS2 without USB voltage regulator	0.05 mA
VLLS3 without USB voltage regulator	0.05 mA

Table 2: Current consumption of different low power modes listed in the data sheet [3, p. 15].

Power mode	Current consumption
I_{DD_VLPR} (4MHz)	1.46 mA
I_{DD_VLPW} (4MHz)	0.61 mA
I_{DD_VLPS}	5.9 µA
I_{DD_LLS}	2.6 µA
I_{DD_VLLS1}	1.47 µA
I_{DD_VLLS2}	1.59 µA
I_{DD_VLLS3}	1.9 µA

3.3.1 Mode Transitions

We do the measurements for the wake-up times from different sleep modes by first entering a sleep mode with the receiving test board and then sending interrupt signals and initiating a I2C communication from the other board. We time how long it takes for the sleeping board to respond and report this. Due to how the I2C message protocol works the receiver needs to be active when receiving the message so the sender has to resend the message until the receiver has fully woken up. This of course assumes that the receiver receives an interrupt which wakes it up.

For reference we measure how long it takes to receive a message when already in run mode. We measure the delay in the response to be 0.1ms. For all other measurements we subtract this from the measured value to get the time it takes to be able to start receiving a message. Notably in the wait mode the time it takes to respond is identical to the time when receiving in run mode. Also in wait mode the receiver is able to respond to the first I2C interrupt, this is not the case for the other sleep modes. The wake-up times we measure might not be exactly equivalent to the

actual processor wake-up time but for the purposes of this experiment the time it takes to respond to a message is more important.

Table 3: Measured wake-up times from different sleep modes.

Power mode transition	Delay
Run → Run	0.00 ms
Wait → Run	0.00 ms
VLPW (2MHz) → Run	0.30 ms
VLPR (2MHz) → Run	0.30 ms
LLS → Run	0.50 ms
VLLS1 → Run	1.65 ms
VLLS2 → Run	1.65 ms
VLLS3 → Run	1.65 ms
LLS without USB voltage regulator → Run	0.50 ms
VLLS1 without USB voltage regulator → Run	1.65 ms
VLLS2 without USB voltage regulator → Run	1.65 ms
VLLS3 without USB voltage regulator → Run	1.65 ms

Table 4: Wake-up times from different sleep modes listed in the data sheet [3, p. 14].

Power mode transition	Delay
LLS → Run	5.9 µs
VLLS1 → Run	112 µs
VLLS2 → Run	74 µs
VLLS3 → Run	73 µs

The data sheet of the MCU lists faster power mode transition times for LLS and VLLSx modes. This could be due to inefficient implementation or differences in the measurement setup. Also in our measurements we are interested in the response time to a message from a sleep mode instead of the actual mode transition. This most likely cause a discrepancy in the wake-up times compared to the data sheet.

3.3.2 Current Consumptions of Different Core Frequencies

The MK20DX256VLH7 data sheet lists the current consumptions at 3mA with 1MHz, 3mA with 2MHz, 4mA with 4MHz, 5mA with 6.25MHz, 8mA with 12.5MHz, 15mA with 25MHz, 23mA with 50MHz, and 31mA with 72MHz with all peripherals clock gates on and in temperature of 25°C [3, p. 17]. The values are listed in table 6. The core frequencies do not exactly match the frequencies we use in the measurements but the current consumptions roughly match the plotted curves as can be seen in figure 5.

Table 5: Current consumption measurements with different core frequencies.

	2MHz	4MHz	8MHz	16MHz	24MHz	48MHz	72MHz	96MHz	120MHz
Run	1.5mA	5.4mA	7.0mA	10.1mA	17.3mA	27.3mA	30.8mA	37.3mA	43.5mA
Wait	1.3mA	5.1mA	6.2mA	8.6mA	13.6mA	19.6mA	20.9mA	25.8mA	30.7mA
Stop	1.3mA	4.2mA	4.8mA	5.8mA	8.7mA	11.1mA	12.6mA	15.3mA	17.9mA

Table 6: Current consumption with different core frequencies based on the data sheet [3, p. 17].

	1MHz	2MHz	4MHz	6.25MHz	12.5MHz	25MHz	50MHz	72MHz
Run	3mA	3mA	4mA	5mA	8mA	15mA	23mA	31mA

It looks plausible that the stop mode in the Snooze library [6] implements a different functionality than the stop mode in the data sheet. In the data sheet the supply current for the stop mode is only 0.35mA under typical conditions whereas the supply current for the stop mode of the Snooze library varies from 1.3mA with 2MHz clock speed to 17.9mA with 120Mz clock speed.

3.3.3 Bus Efficiency

It is not feasible to measure the time it takes to transmit a message on a given core frequency for all message lengths so we need a model for the time of transmission. For the model for the transmission we assume that for a given core frequency there is a constant time that does not depend on the message length, we call this $BusOverhead$. This overhead could be caused for example by the transmission of the message header or the initiation of the communication. The actual message transmission has some transmission speed, we call it $BusSpeed$.

Using this model we calculate the total time $TotalMessageTime$ it takes to transmit a message of length $MessageLength$ with the following formula:

$$TotalMessageTime = \frac{MessageLength}{BusSpeed} + BusOverhead \tag{13}$$

Only $TotalMessageTime$ can be measured so in order to calculate $BusSpeed$ and $BusOverhead$ we need two sets of measurements of $TotalMessageTime$ with different message length $MessageLength$. Using these two measurements we can solve $BusSpeed$ and $BusOverhead$ from function 13.

$$T_1 = \frac{M_1}{V_B} + T_B$$

$$T_2 = \frac{M_2}{V_B} + T_B$$

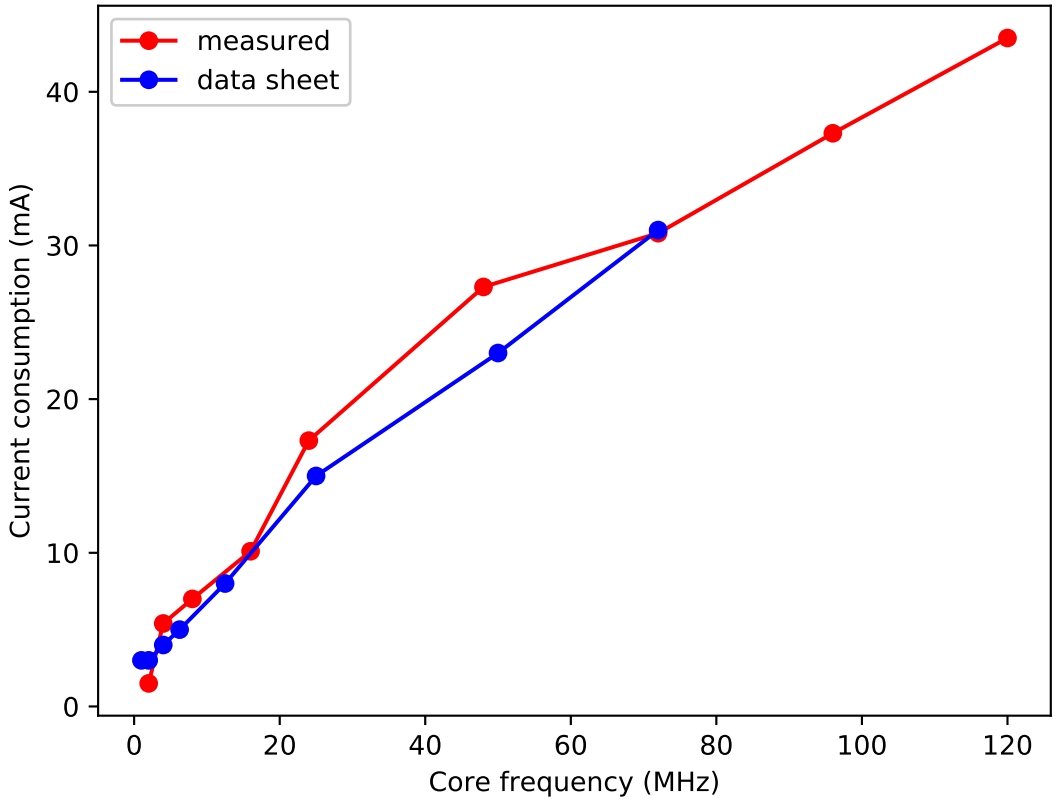

Figure 5: Current consumptions based on the data sheet and measured values in run mode. The graph of the data sheet values is adapted from the K20 Sub-Family Data Sheet [3, p. 17]

Where T_1 and T_2 are the times it takes to transmit the messages of length M_1 and M_2 respectively. V_B is the bus speed and T_B is the bus overhead. With some simple algebra we can rearrange the equations to solve V_B and T_B.

$$V_B = \frac{M_1 - M_2}{T_1 - T_2}, T_B = \frac{M_1 \cdot T_2 - M_2 \cdot T_1}{M_1 - M_2}, M_1 \neq M_2, T_1 \neq T_2$$

For the calculations of the actual bus speed and bus overhead we do one set of measurements with 40 times a 250 byte message and another with 400 times a 25 byte message. We measure the total time for each set of transmissions and divide the measurement by the number of messages to get the duration of sending one message. We do this to reduce the effect of variance in the message transfer. We present the resulting bus speeds and bus overheads for any core frequency in table 7.

The maximum baud rate for the I2C device is clock/20 [4, p. 1169]. This means with 2MHz core frequency the I2C bus speed is limited to 100kbit/s, similarly with 4MHz the bus speed is limited to 200kbit/s. For other clock speeds the bus speed

Table 7: Speed of the I2C bus with different core frequencies.

	2MHz	4MHz	8MHz	16MHz	24MHz	48MHz	72MHz	96MHz	120MHz
I2C 250bytes (bytes/s)	4130	9200	18300	25100	28300	31800	32100	32200	32200
I2C 25bytes (bytes/s)	3700	8190	16700	22200	24700	27500	27800	27900	27900
V_B (bytes/s)	4180	9330	18500	25500	28800	32400	32700	32800	32800
V_T (ms)	0.782	0.371	0.145	0.145	0.143	0.137	0.134	0.133	0.133

is limited to 400kbit/s in this experiment as it is the maximum speed of the first standardized I2C specification [2]. Higher bus speeds could be used but it would require the master nodes to support higher bus frequencies which is not always the case.

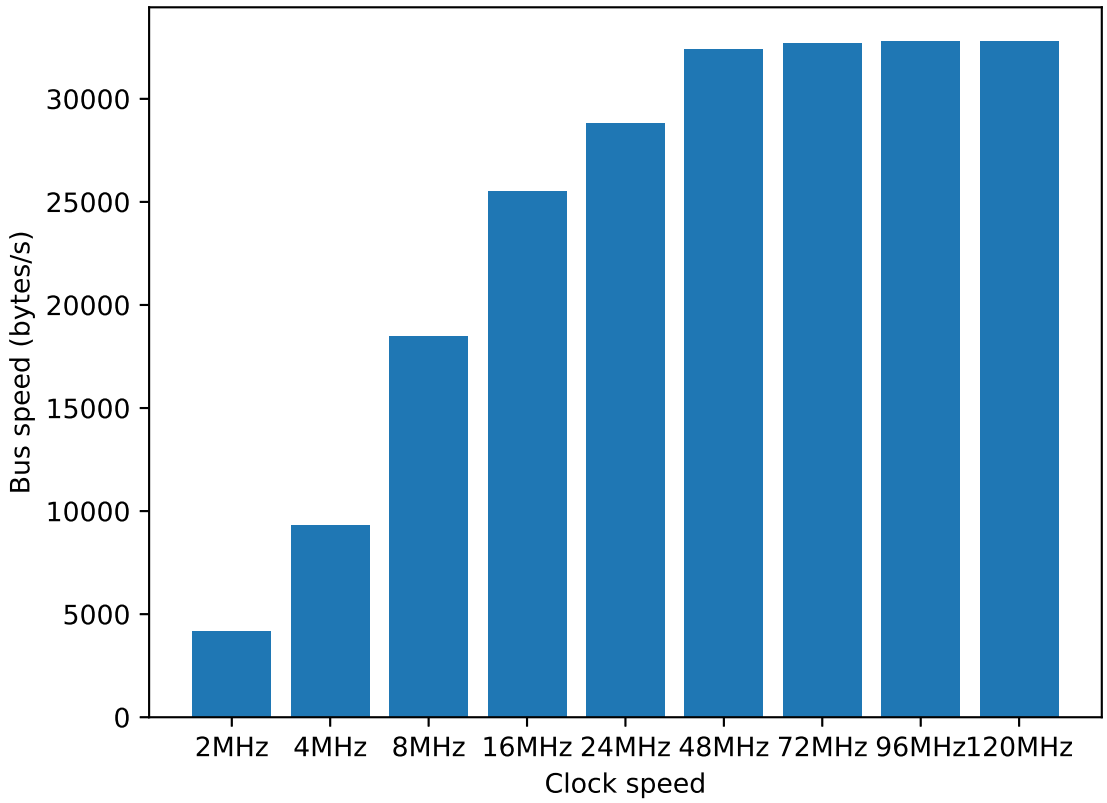

Figure 6: Bus speed at different clock frequencies.

We can observe from figure 6 that the measured bus speed increases steadily from

2 MHz core frequency to 8 MHz along with the maximum bus baud rate. After 16 MHz the bus speed starts to suffer from diminishing returns as the measured bus speed is limited by the maximum bus baud rate. Also the core frequency of the sending device may affect the measured bus speed; in this experiment the sending device uses a core frequency of 48 MHz. We still observe some speedup due to the fact that the transmission requires some processing which is not limited by the bus baud rate. The bus speed approximately follows Amdahl's law [11]. This is because we can assume that the transmission has two independent tasks, the actual transmission through the bus and the processing required for the transmission. The transmission speed through the bus stays constant after 8 MHz whereas the message processing can be sped up.

We can use the measured bus speeds with the current consumptions to calculate the efficiency of the message transmission. We calculate he efficiency with:

$$Efficiency_{BUS} = \frac{BusSpeed}{I_{RUN}} \tag{14}$$

Where $Efficiency_{BUS}$ is the efficiency of the bus and I_{RUN} is the current consumption of the processor on a given core frequency. The calculated bus efficiencies can be seen in table 8 and figure 7.

Table 8: Bus efficiency with different clock frequencies.

	2MHz	4MHz	8MHz	16MHz	24MHz	48MHz	72MHz	96MHz	120MHz
Bus speed (bytes/s)	4180	9330	18500	25500	28800	32400	32700	32800	32800
Bus Efficiency (bytes/mAs)	2790	1730	2640	2520	1660	1190	1060	878	753

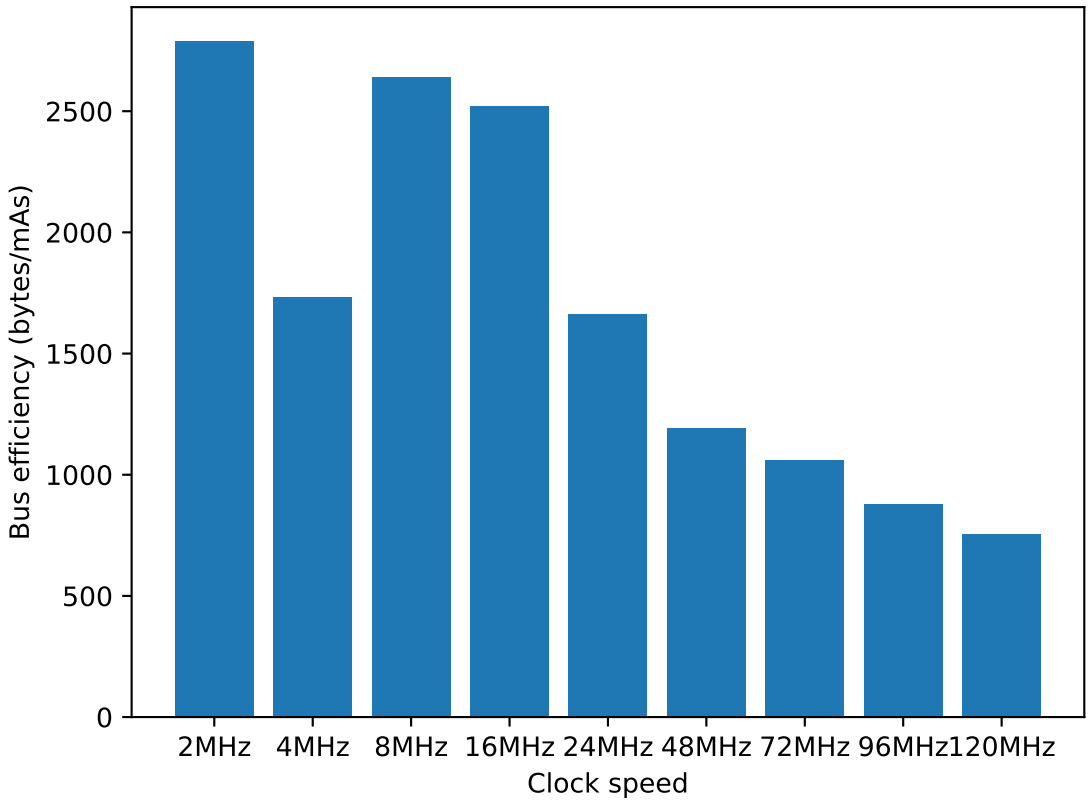

Figure 7: Bus efficiency at different clock frequencies.

4 Power Management

In this chapter we explain the communication model between the communicating devices. We investigate the effect of different processor characteristics in terms of the power consumption. We explore different types of power managers and we present and analyze algorithms for calculating the power consumption with different power managers.

4.1 Message Communication Model

For the communication model we assume there is a single device that is connected to many external devices. For convenience we refer to this single device as the receiver as it receives transmission sent by the external devices. Likewise we refer to the external devices as senders.

We assume that no optimization can be done in the external devices. The external devices work as is. Only their power consumption, transmission length and transmission period is known. The power consumption is some constant when transmitting or when waiting for the receiver to accept a transmission. During inactive periods the senders have a power consumption of zero. All the external devices have a communication channel with the receiver. The external devices are not connected with each other so all communication goes through the receiver.

Any of the external devices can initiate a transmission at any point. Only the external devices can initiate a transmission with the receiver, the receiver only receives incoming connections and never initiates them. In other words it is a configuration where the receiver behaves as a slave and all the senders behave as masters. It is possible for a sender to request the receiver for data but the external device has to always initiate this. For the purposes of this model the two-way communication behaves no differently from the sender simply sending a single message. For simplicity we assume that all transmissions are a single transmission from a sender to the receiver, a transmission which the receiver acknowledges. Also we assume the transmission speed is not affected by the external device meaning the sender can provide whatever speed the receiver uses.

The main goal for the power manager in this section is to minimize to power consumption while having no real time requirements or additional processing requirements for the message handling. Delaying a transmission has a linear power consumption loss in terms of the time delayed so minimizing the power consumption automatically prefers not delaying transmissions indefinitely.

Depending on the characteristics of the system different power management configurations are the most suitable for a given system. The following parameters affect the power management configuration:

1. Power consumption of different modes of the receiving device
2. Available modes of operation in the receiving device
3. Transition times between different modes of operation

4. Power consumption of the transmitting devices
5. Periods and lengths of transmitted messages

4.2 Power Manager

Table 9: Parameters affecting the total power consumption of a given system.

M	Set of all processor modes
m	Processor mode $m \in M$
P_m	Power consumption in mode m
P_{EXT}	Power consumption of the external transmitting device
$V_{B,m}$	Bus speed in mode m
$T_{B,m}$	Bus overhead in mode m
$t_{m,n}$	Transition time between modes m and n
$P_{m,n}$	Power consumption of the transition between modes m and n
A	Set of all periodic messages
T_a	Period of message a
l_a	Length of message a

The set of processor modes is a set of all possible processor configurations with distinct characteristics in terms power consumption and message transmission speed. In this thesis, however, the processor modes are limited to different core frequencies as it has the greatest impact on the message transmission speed and power consumption.

For this purpose $P_{EXT} = P_{ACTIVE}$ and $P_{INACTIVE} = 0$ for the external device. We assume that the inactive power consumption of the external device zero. This simplifies the calculations and the power consumption in the sleep mode is close to zero in most cases.

The processor modes can be divided into inactive and active modes. In active modes messages can be received and processed. In inactive modes no messages can be processed but the power consumption is lower. This divide can be observed easily; naturally all modes that cannot process messages are inactive modes. Furthermore every active mode has a respective idle mode where the power consumption is lower but no messages can be processed. The transfer between a run mode and its respective idle mode is near instantaneous. Such mode can always be generated simply by copying the characteristics of the active mode with the exception of setting the message processing speed to 0. Every inactive mode can transition into every active mode and vice versa which means the power consumption modes form a complete bipartite graph (illustrated in figure 8). This simplifies the calculation for which transitions should be made given a message system as transition can only happen between an active and an inactive mode. Since the fastest transfer between an active mode and an inactive mode is instantaneous no power is ever lost in the calculation by doing a transfer into a inactive mode between two active sections.

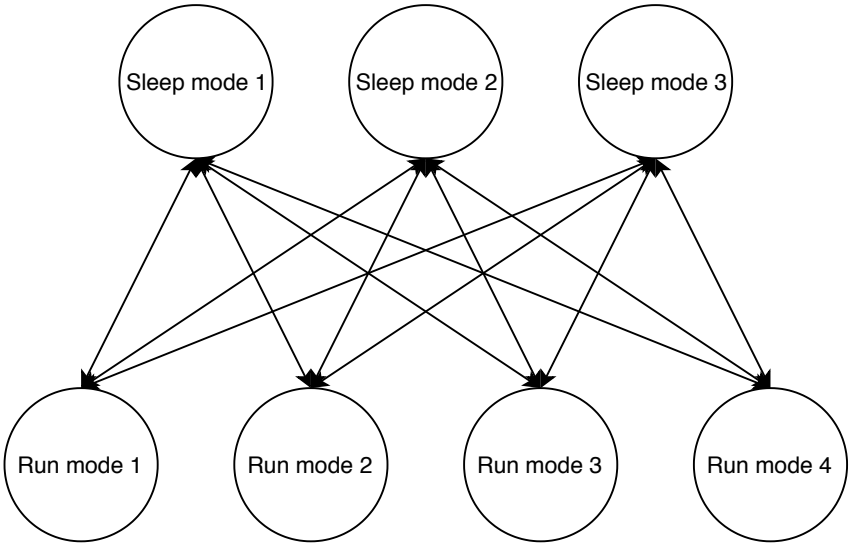

Figure 8: Power mode transitions in a system with 3 sleep modes and 4 run modes.

Calculating the power consumption for systems with a single external device is fairly simple.

Time it takes to transmit message a in mode m:

$$t_{a,m} = T_{B,m} + \frac{l_a}{V_{B,m}} \tag{15}$$

Energy consumption of a transmission of message a in mode m:

$$E_{a,m} = (P_m + P_{EXT}) \cdot t_{a,m} \tag{16}$$

$$E_{a,m} = (P_m + P_{EXT}) \cdot (T_{B,m} + \frac{l_a}{V_{B,m}}) \tag{17}$$

Calculating the most efficient core frequency for a given message length a and power consumption of the external device P_{EXT}:

$$E_{a,m_{min}} = \min_{m \in M}((P_m + P_{EXT}) \cdot t_{a,m}) \tag{18}$$

The most efficient core frequency for a given message of length a and current consumption P_{EXT} of the external device calculated with the values gotten from table 7 are shown in figures 9 and 10. Each color represents the area where that core frequency is optimal from the set of all core frequencies calculated. Note that some core frequencies are not optimal for any combination of sender power consumption and message length in the range of a and P_{EXT} given in the figures.

Average power consumption of a periodic message a transfer in mode m and waiting in mode n assuming wake-up time is 0.

$$P_{total} = \frac{(P_m + P_{EXT}) \cdot t_{a,m} + (T_a - t_{a,m}) \cdot P_n}{T_a}, T_a \geq t_{a,m} \tag{19}$$

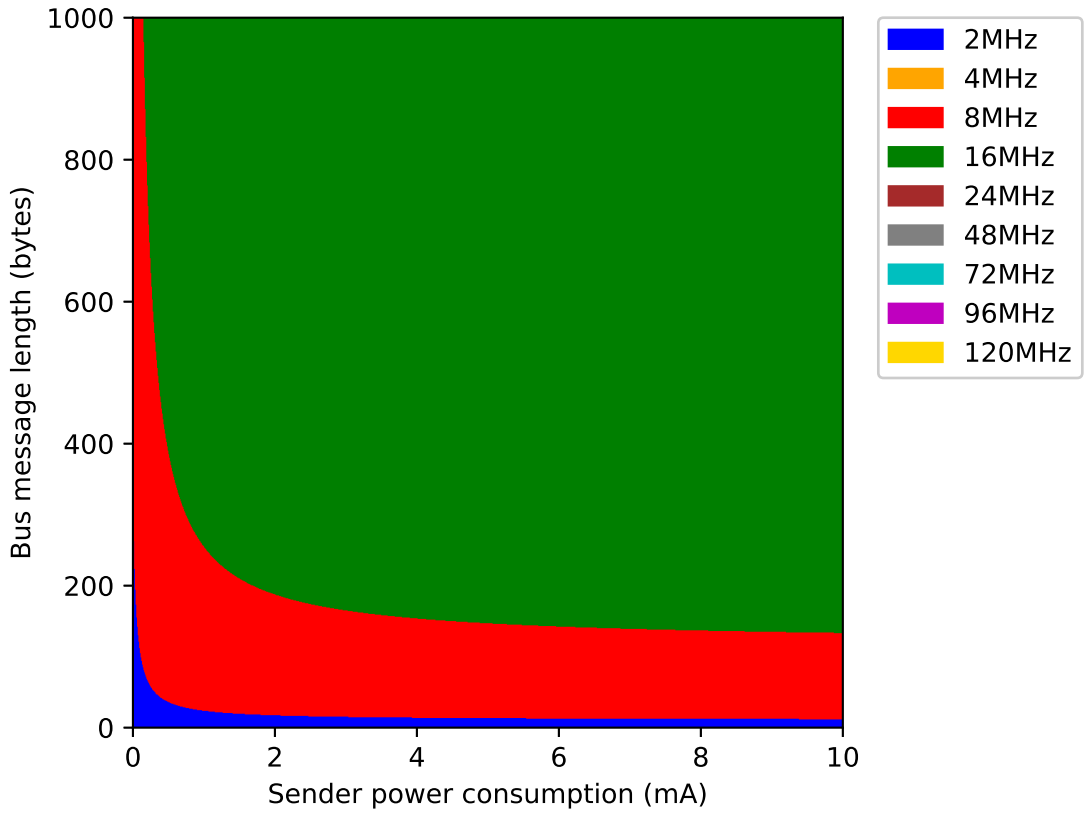

Figure 9: Most efficient core frequency for a given external power consumption and message length.

Adding the wake-up time to the formula requires the transition time from waiting mode m to running mode n.

$$t_{a,m,n} = T_{B,m} + \frac{l_a}{V_{B,m}} + t_{m,n} \tag{20}$$

Average power consumption of a periodic message a transfer in mode m and waiting in mode n including the wake-up time.

$$P_{total} = \frac{(P_m + P_{EXT}) \cdot t_{a,m,n} + P_n \cdot (T_a - t_{a,m,n})}{T_a}, T_a \geq t_{a,m,n} \tag{21}$$

For a setup with two or more external transmitting devices it is considerably more difficult to write the average power consumption in closed form as the incoming message transmissions can overlap with each other. This means that one of the external devices might have to wait for the previous transmission to complete before being able to start the transmission. Also it is not always optimal to enter into a sleep mode between messages if the next message arrives soon enough after the previous message. Also in this calculation we make the assumption of having the wake-up be during the transmission which does not necessarily have to be the case. Starting the wake-up sequence at the transmission initiation is one approach of how

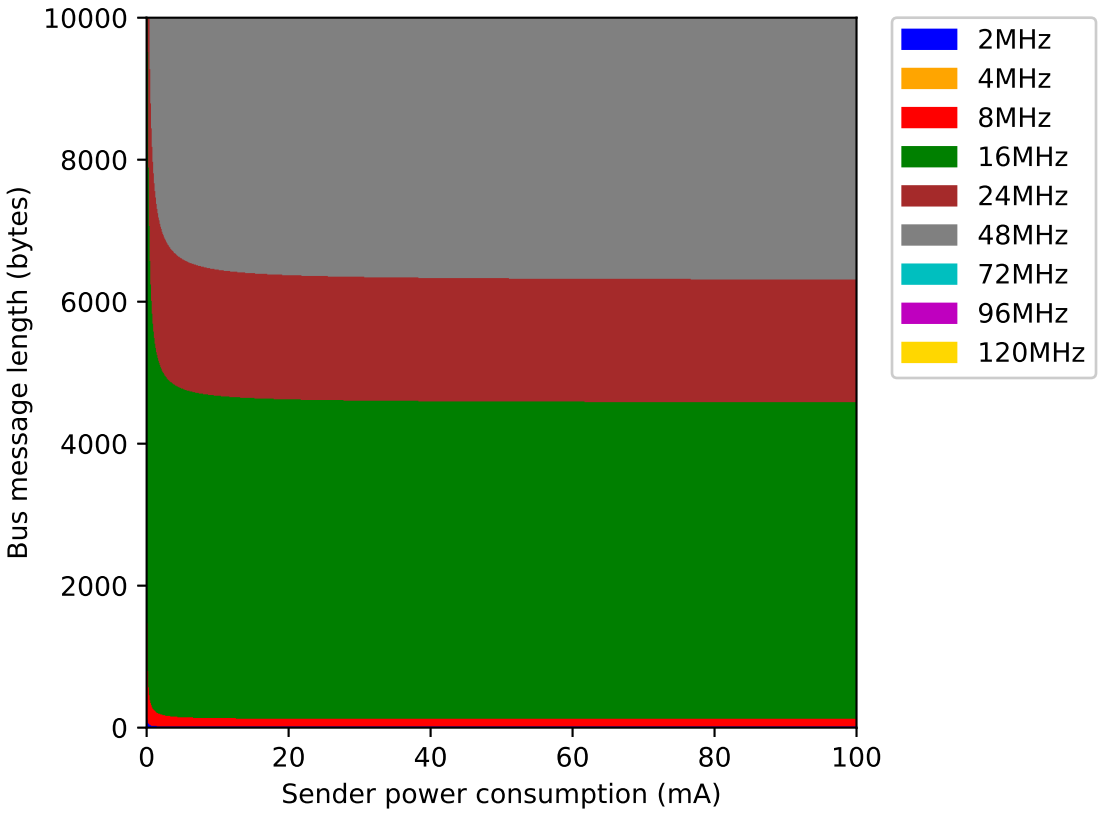

Figure 10: Most efficient core frequency for a given external power consumption and message length.

to handle the waking up but it is not the only one. We explore the other wake-up schemes in a later segment.

If the transmitters use separate communication busses (illustrated in figure 11) many of the well-established scheduling algorithms would work as the power manager for a configuration like this. One option is the shortest remaining time algorithm. With this algorithm the tasks are placed in a priority queue. Whenever a task finishes or a new task arrives the queue will be searched for the task with the estimated shortest time remaining. Because the shortest task is always processed first the average waiting time for the tasks is usually shorter. However, in a system with a lot of small tasks this algorithm would cause a lot of context switches and as a result poorer efficiency.

Another option is a fixed priority scheduling algorithm. In this algorithm each task is assigned a fixed priority and the tasks are processed based on these priorities. Low-priority tasks get interrupted if a high-priority task arrives and the high-priority task is processed instead. This algorithm can be efficient in systems with different power consumptions for each task. The high power consumption tasks could have higher priority and thus get processed first. Other scheduling algorithms also exist but these are not explored further in this thesis.

The transmitters could also use a common bus (illustrated in figure 12). In this configuration we assume that if the common bus is currently in use no other node can access the communication bus i.e. the transmission is blocking. This severely limits the options for scheduling algorithms as the receiver cannot know which transmissions are in the queue and we would no longer be able to use the scheduling algorithms presented in the previous paragraphs. If every transmission is accepted the scheduling scheme would follow a simple FIFO (first in, first out) algorithm. Delaying the reception of a transmission in favor of receiving a different transmission would increase the options for scheduling. For this to be advantageous the receiver would need to have very accurate predictions of the transmissions being sent. The receiver cannot be ensured that another transmission is waiting for the bus so it would need to be able to preempt it. Also depending on how the transmitting devices behave the same device might be the first one to reconnect after just having its transmission rejected. Avoiding this problem would require additional logic on the side of the sender.

Context switching also suffers from the same issues as delaying the receptions of a transmission; the receiver would need to know that another transmitter is waiting for the bus to become available. However, if these issues could be solved the same algorithms could be used as in the system with separate communication busses.

The advantage of having only one communication bus is that every device can use the same bus and thus removing complexity and simplifying the hardware configuration. With the common communication bus the system is easily extendible with additional devices instead of requiring a separate connection for each device. This is the type of configuration we examine in this section.

We assume that the receiver has enough time to process every message i.e.:

$$\sum_{a \in A} t_{a,m_{max}} \leq T_a, \text{where } t_{a,m_{max}} = min(t_{a,m}), m \in M \tag{22}$$

If this is not the case it means there is not enough time to process all messages so some messages would need to be rejected based on some criteria. A different power management algorithm would need to be used for this for example the algorithm proposed by Rusu [27].

If there is a requirement that every transmission needs to be processed before the next transmission from the same node additional constraints would be needed. Power management algorithms with hard deadlines are a well-researched area for example by Liu [21].These additional constraints are not in the scope of this thesis.

The following settings determine the power consumption of a given system:

1. Core frequencies used
2. Sleep modes used
3. Conditions for going to sleep
4. Conditions for waking up

The different power managers vary on how each of these is decided.

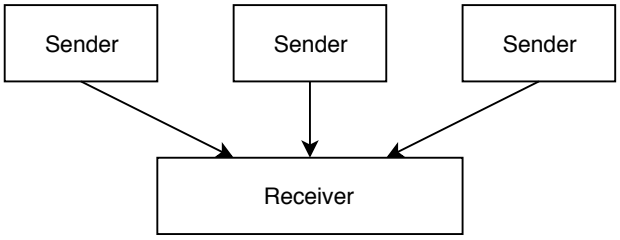

Figure 11: The topology of multiple transmitters with separate communication busses.

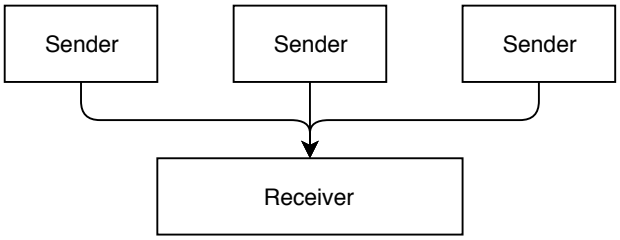

Figure 12: The topology of multiple transmitters with a single communication bus.

4.3 Core Frequency

The core frequency the system uses has a large impact on the total performance and power consumption of the system as the bus throughput and power consumption vary greatly across different core frequencies. It is possible to change between different processor core frequencies, and it could be advantageous depending on the distribution of the intensity of incoming transmissions.

During a transmission there is always another device communicating with the receiving device. Naturally the external device has to be active during the transmission and the time it needs to stay awake affects the total power consumption of the system. This of course assumes that the external device has some sort of low power mode it can enter. This means the bus efficiency of the receiver is not the only aspect that matters.

4.4 Sleeping Schemes

The selection of sleeping schemes for the power manager is very important. Different sleeping schemes are possible and they affect the characteristics of the system greatly.

The simplest scheme is to never use any sleep modes other than the shallowest sleep mode that has an instant wake-up. This has the advantage of being very simple to implement; there are no additional functionalities needed in the software or hardware. It also has the best response time and handles irregular communications effectively. However, the power consumption is high also on systems with infrequent communications. This is because the processor stays constantly in a state with high power consumption. Even so this scheme can be the best option for high throughput or time critical configurations.

Always entering a sleep mode when not active is another simple option. This would mean that the receiver enters a sleep mode after a message transmission is completed and when no other message or other task is waiting. This is also fairly easy to implement as it works the same way regardless of the message scheme. It is also effective on systems with rare message transfers with a large proportion of inactive time. It, however, always has slow response times to messages since it always needs to wake up first before receiving the message. It can also fare poorly when transmissions happen frequently as seen in figure 13. Note that in this example no time is actually spent sleeping and the receiving of the message is delayed due to the enter sleep and wake up cycles.

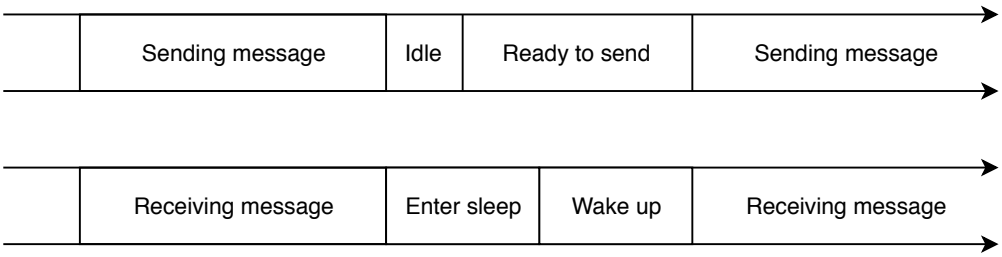

Figure 13: Example of an ineffective sleep cycle.

Switching to a lower power mode during idle time is another option. This mode works by lowering the core frequency to some lower state. This means a lower power consumption than in the full power mode while still retaining an instant response to the beginning of a message transfer. For this to be utilized a more sophisticated sleeping scheme needs to be in place. It could be a good choice for example to a system with regular message transmissions with unpredictable but time critical transmissions.

If the address of the sender is known by the receiver, like in the I2C message protocol, the address can be used to decide whether to enter sleep mode after the message is received. This can be more efficient on some systems, for example consider a system with two senders, one sending regular messages to a receiver and another sender sending messages twice as long but half as often (shown in figure 14). After the longer message only a shorter idle period can occur whereas after the shorter message both longer and shorter idle periods can occur. So if the weights of the sleep and idle modes are suitable after the longer message the receiver should enter an idle mode whereas after the shorter message sleep mode is more appropriate. An ever more pronounced difference is seen with messages with same period but with an offset. Transmission based sleeping scheme of course requires that we know some characteristics about the frequency and lengths of messages sent by different addresses.

Then there is the universal optimal sleeping scheme which assumes that every message length and time of transmission is precisely known. The receiver is able to predict the next instance of a message transmission precisely and therefore is able to make the optimal decision which mode of operation to use. This requires that all the messages are sent on schedule with no irregularities. Also for any larger message

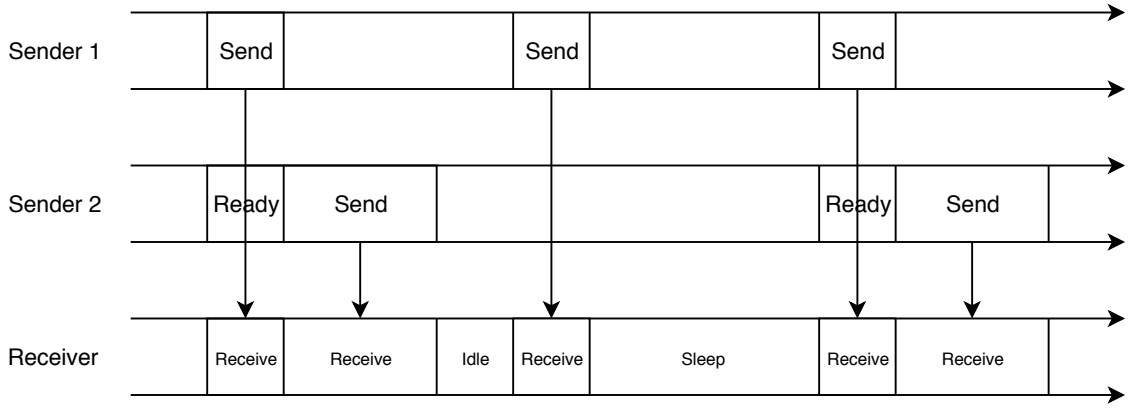

Figure 14: Example of a message system with improved efficiency with address based sleep.

system the table of mode transitions may grow very large. This scheme might not be very practical for most systems but it can be useful for calculating the theoretical minimum for the power consumption for a given system.

4.5 Wake-up Schemes

The selection for wake-up schemes for the power manager is narrower than the selection for sleeping schemes. The options are no sleep, interrupt based wake-up, and preemptive wake-up. No sleep is straightforward; if the receiver never sleeps it never needs to wake up. In an interrupt based wake-up scheme the sender of the message wakes the receiver either through one of the message lines or some other communication line by sending a wake-up signal. The advantage of this is scheme is that the receiver only wakes up when a message is being sent. The downside is that the receiver does not respond instantly and thus the senders need some additional logic to handle the wake-up signal sending and the delay between sending the message and getting a response. Depending on what type of message transmission the sender uses it may either need to wait for the receiver to be ready or send the message multiple times until the message is received successfully.

The third wake-up scheme is a preemptive wake-up. The receiver wakes up just before a message is about to be sent. This allows the sender to start the transmission immediately without having to wait for the receiver to wake up. However, for this scheme to be effective the receiver needs to know the timing of the message fairly accurately in advance, otherwise it would either wake up too early and waste power being active or wake up too late after the sender has already started transmitting. This is also the theoretically optimal wake-up scheme. The receiver would wake up so that it could receive a message immediately while neither it nor the sender needs to wait for the other to act.

Also a combination of the interrupt based and preemptive wake-up schemes could be used. The interrupt would act as a backup when the preemptive wake-up is not

triggered.

4.6 Message Schedule

For the general case where we have a list $\{r_1, r_2, ...\} \in R$ where R is a list of transmissions between a sender and the receiver. Each r_i has length l_i, start time t_i, and power consumption p_i. We calculate the energy required for completing an arbitrary list of transmissions with function 23 where $transmission(r_i)$ is the energy consumed from the start of the transmission to the start of the next transmission. For simplicity if $r_i.t = r_{i+1}.t$ i.e. two messages start at the same time r_i is processed first. This means the second transmission waits until the first transmission is completed.

$$E_{total} = \sum_{r_i \in R} transmission(r_i) \tag{23}$$

4.7 Recurring Message Scheme

We have a list $\{d_1, d_2, ...\} \in D$ where D is a list of devices connected to the receiver. Each device has some recurring message transmission a with sending period of T_a, message length of l_a and an external power consumption $P_{d_i,EXT}$. From these we create the list of transmissions by taking every device d_i and generating all multiples of T_a under the period where the messages start to repeat, we call this period where all messages repeat the hyperperiod. The hyperperiod which we need to investigate is $lcm(T_1, ..., T_A)$ where lcm is the least common multiple. After a segment of this length the cycle repeats. This is easy to see as time $t = lcm(T_1, ..., T_A)$ is divisible by each of the periods meaning the message schedule is the same as at time $t = 0$. This of course means that we need to ensure that the message transfers don't overflow to the next hyperperiod i.e. the processing of the last transmission does not overlap with first message transfer of the next round of transmissions. For recurring messages it is a good heuristic to assume that if at time $t = lcm(T_1, ..., T_A)$ the previous message is overflowing, the message cycle is unprocessable. However, this does not always mean that the cycle is unprocessable at least in the general case. With an arbitrary transmission schedule we can easily create such a case, for example by adding a transmission to $t = lcm(T_1, ..., T_A) - \varepsilon$ to some sparse transmission schedule.

With this procedure we generate a list of transmissions R. The lists of transmissions generated using this procedure is a subset of all possible list of transmissions and all algorithms applicable for the general case are also applicable for this special case.

4.8 Optimal Power Manager

In this section we investigate the optimal power manager in more detail, a power manager with pre-emptive wake-ups and an optimal sleeping scheme. We use a FIFO method in the ordering of the transmissions meaning incoming messages are received in the order of arrival. If two messages arrive at the same time the message from the device with a lower index arrives first. It can be more efficient to delay receiving of

certain messages for example delaying a long but low energy transmission in favor of a shorter but more energy intensive transmission. However, this would complicate the message receiving logic. Also we assume messages will always be received in their entirety so no interrupts occur from other devices in the middle of transmissions.

We solve the optimal mode transitions for an arbitrary message scheme R by the following algorithm. Treat each transmission as a level of nodes and each node as a power mode. Between two levels of run modes there is a level of sleep modes. Set directed edges from all nodes to the nodes in the level above (an example tree shown in figure 15). The edges between the nodes are transitions between power modes. In other words the problem can be represented as a directed acyclic graph. The transitions from a run mode to another run mode is a set of all different sleep modes during the transition (figure 16).

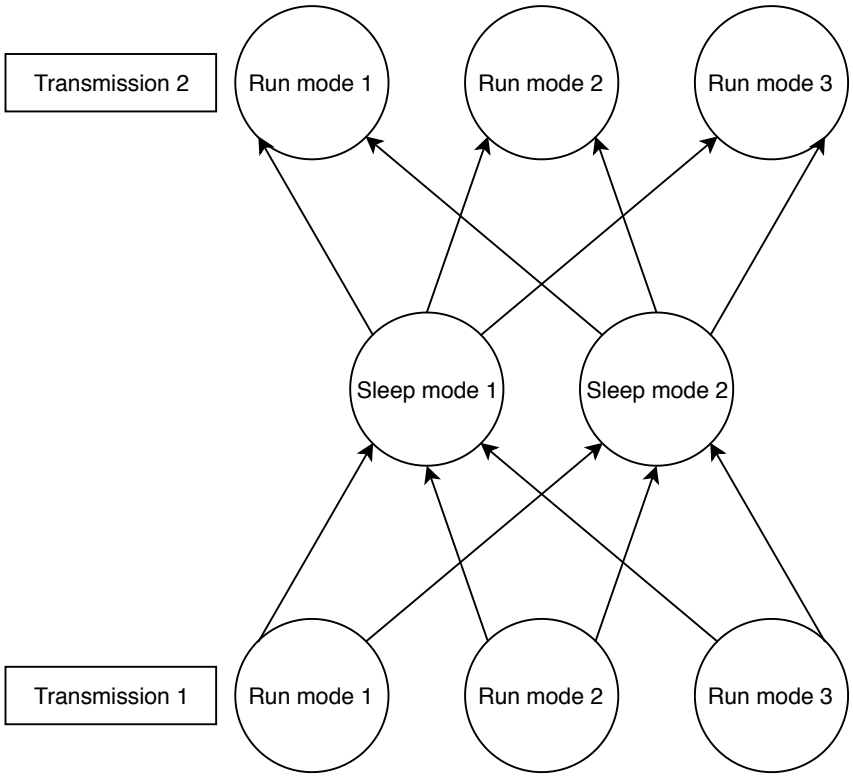

Figure 15: A 2-transmission schedule represented as a tree.

Each node is a tuple of $(time, energy)$. Each edge has some duration t and some energy consumption E. The goal is finding the shortest path in relation to time and energy from the first level of nodes to the last level of nodes. The solution will be some set of tuples of time and energy. This is because it is ambiguous which is better between pairs of different $(time, energy)$. Also the whole graph cannot be created in advance because the weights between nodes can change depending on when the transmission starts and how long it takes for example whether a transmission overlaps the start of the next transmission. This would mean the power consumption of the device of the second transmission is also relevant.

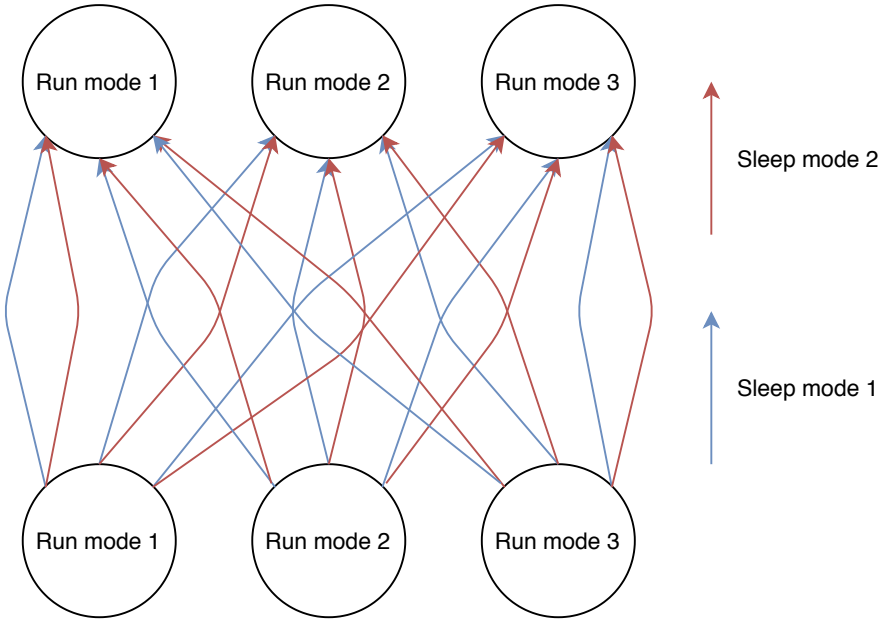

Figure 16: A 2-transmission schedule created from figure 15 represented as a directed acyclic graph with sleep modes as directed edges.

Using topological sorting the shortest path problem for directed acyclic graphs can be solved in linear time relative to the number of vertices and edges i.e. $\mathcal{O}(E+V)$. Due to the topology of the graph a simple breadth-first search will be in topological order. No matter the path taken the distance from one node to another node is always the same. Because of this for all edges uv the vertex u will come before v in the breadth-first search ordering. Since every vertex has only one incoming edge i.e. $V = E$ the shortest path for this problem can be solved in $\mathcal{O}(V)$ time. However, the number of vertices grows exponentially in relation to the number of transmissions. The naive calculation of traversing through every potential path combination would require $\mathcal{O}(M^R)$ operations where M is the number of modes and R is the number of transmissions. The problem is that multiple values of $(time, energy)$ tuples need to be stored for each node and each of them needs to be treated as a potential candidate for the shortest path. This leads to the exponential growth in the number of nodes. Using a more sophisticated algorithm the average case time complexity can be brought down significantly.

We use a variation of breadth first search to calculate the optimal path more efficiently. The variation comes from the fact that both $time$ and $energy$ are relevant and thus most of the time it is not possible to prune all nodes with the same mode in a level to only one entry. We have to keep a list of different $(time, energy)$ combinations in store for each mode. We can, however, prune some of the nodes out. We only need to keep the members of the Pareto frontier for the next iteration. That is when $(time, energy)$ tuples U and V if $U.time > V.time$, $U.energy > V.energy$ and $U.mode = V.mode$ (figure 17). In other words all nodes that are strictly dominated by another node with the same mode can be pruned out. The pruning is done with algorithm 1 and the full algorithm for calculating the shortest path is given

by algorithm 2. Disser [17] gives an alternative implementation for multi-criteria shortest path for directed graphs.

It is possible that nothing gets pruned out but if the entries are uniformly distributed most get pruned out. For a uniformly distributed group the expected number of members left after the pruning is the n-th harmonic number H_n where n is the number of entries. This can be easily seen after the list of entries is sorted by the first value. Then iterating through the list and if the second value of an entry is smaller than all the previous values the entry is kept. For the first entry the chance is 1, for the second $\frac{1}{2}$, for the third $\frac{1}{3}$, and so on. The harmonic series can be approximated by the natural logarithm $\ln(n)$ as $H_n < \ln(n) + 1$. The number of nodes on nth level is calculated with function 24 where S is the number of sleep modes and M is the number of run modes.

$$x_n = \ln(x_{n-1} \cdot S \cdot M) \tag{24}$$

Also the function 24 has a solution where $x_n \geq 1$ meaning as n approaches infinity, $x_n = x_{n-1}$ which in turn means that the number of edges between two levels of nodes is a constant when S and M are constants and when n approaches infinity. If we assume this the time complexity is linear in terms of the number of transmissions.

There is an edge case where the algorithm might fail to find the optimal solution. The optimal subpath might get pruned out if for nodes U and V, $U.time > V.time$, $U.energy > V.energy$, $U.mode = V.mode$ and $(U.time - V.time) \cdot U.sleep > U.energy - V.energy$ where $U.sleep$ is the power consumption of $U.mode$. This is because the pruning assumes that finishing a transmission earlier is always better which allows for the next transmission to be started sooner. However, if the external power consumption of the device is very low, starting the next transmission earlier might give no advantage. Since it is not possible to advance in time without consuming some amount of power a Pareto suboptimal point can become an optimal candidate after some number of transmissions.

All the previous calculations assume that the entries are uniformly spread. It is possible that this is not the case so we cannot assume a result this strong for the algorithm 1 for the worst case. But the simulations we do in the following chapter seem to suggest this result. For a relatively large number of transmissions the algorithm finds the optimal schedule in a reasonable amount of time.

4.9 Optimal Power Manager with Interrupts

The transition times between different processor modes are so short that it can be very difficult to wake-up preemptively and be accurate enough to get the advantage of not needing to wait with the sender for the receiver to wake up. Because of this the interrupt based power manager is also relevant.

The algorithm for this is very similar to the case with pre-emptive wake-ups. The only difference is that the time to wake up is included in the run node. We assume the receiver wakes up when the transmission starts and starts receiving the transmission when the waking sequence is over. During the wake-up period the sender still consumes power.

Algorithm 1 Pareto prune

```
    Input: List of (time, energy) tuples for a node with the same power mode
 1: function PruneEntries(candidates)
 2:     candidates.SortBy(time)
 3:     pruned ← []
 4:     for each entry in candidates do
 5:         if entry.energy < Last(pruned).energy then
 6:             pruned.append(entry)
 7:         end if
 8:     end for
 9:     return pruned
10: end function
```

Algorithm 2 Most efficient path search

```
    Input: A graph of all transmissions with possible power modes for each
 1: function OptimalPath(graph)
 2:     for each level in graph do
 3:         for each node u in level do
 4:             u ← PruneEntries(u)
 5:             for each entry w in u do
 6:                 for each outgoing edge from u to v do
 7:                     newEntry ←(w.time+time(w, v), w.energy+energy(w, v))
 8:                     v.addEntry(newEntry)
 9:                 end for
10:             end for
11:         end for
12:     end for
13:     result ← []
14:     for Each node n in Last(level) do
15:         result.add(n)
16:     end for
17:     result ←PruneEntries(result)
18:     return result
19: end function
```

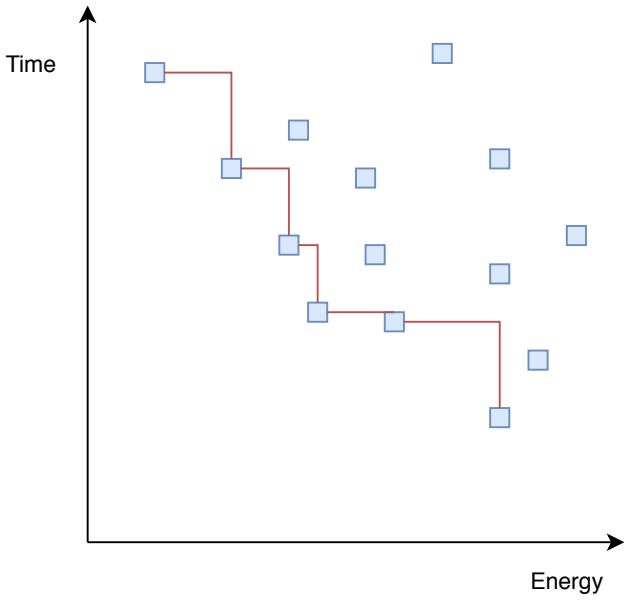

Figure 17: The pareto frontier in an example node.

4.10 Sender Based Power Manager

In the sender based power manager the receiver selects the sleep mode to enter based on the address of the previously received message. After generating a list of transmissions $\{r_1, r_2, ...\} \in R$ using the list of devices $\{d_1, d_2, ...\} \in D$ each transmission r_i has a device that sends it. Let us call the transmission sent by device d_j as $r_i.d_j$. Let us call the inactive mode after $r_i.d_j$ $d_j.sleep$. In order to find the best configuration the lowest power consumption setup needs to be found for all $d_j.sleep$ where $d_j \in D$ and $sleep \in S$ where S is the set of all sleep modes. To calculate the best sleep mode for each device we need to calculate the total power consumption of the hyperperiod with function 23 with all the combinations of sleep modes for all d_j. In the case where the end of a transmission occurs simultaneously with a start of another transmission we assume that the receiver does not go to sleep after the first message has finished sending but instead receives the second message immediately after the first has finished. Implementation wise this is not problematic because the receiver is active for a moment before going to sleep.

This scheme also assumes that the core frequency stays the same across the hyperperiod of transmissions. Changing the core frequency is power consuming and this scheme cannot anticipate segments of high activity and would unnecessarily change the core frequency often. The simple case of one run mode and one sleep mode across all transmissions is one configuration of this scheme, more specifically where $d_j.sleep = d_i.sleep$ for all $d_j \in D$ and $d_i \in D$.

4.11 Predictive Power Manager

In the predictive power manager at the end of each transmission a decision is made whether to enter a sleep mode or stay in idle mode. This is based on how long it takes until the next transmission starts. Since we know when the next transmission starts we can calculate the threshold for entering a sleep mode with equation 25. We compare the power used in the sleep mode to the power used in the idle mode. If the equation is true sleep mode should be entered. This does not necessarily give the most optimal answer every time, it is possible that delaying the next message delays the message after it and so forth and as a result causes a higher power consumption.

$$t_{NEXTTRANSMISSION} < \frac{t_{WAKEUP} \cdot (P_{EXT} + P_{RUN})}{P_{IDLE} - P_{SLEEP}} \tag{25}$$

Naturally it would be possible to calculate further ahead and be able to make a more accurate prediction which inactive mode to enter but that would increase the complexity and the amount of computation needed in the receiver. Also calculating any constant number of transmissions ahead does not guarantee that the best decision will be found. In order to guarantee it potentially all of the transmissions in the hyperperiod would need to be considered.

An advantage of this power manager is that it could also be a reasonable for message schemes with uncertainty or previously unknown message schedules. These of course would need a different type of predictor, something that logs the previous inactive cycles and predicts accordingly would be appropriate.

5 Simulations

Measurements gotten in the measurements chapter is used in all the simulations in this chapter. We only need to include some of the sleep modes in the simulations because some of the sleep modes are strictly dominated by another sleep mode in terms of both wake-up time and current consumption. We only need to consider VLPR, VLPW, and LLS without USB voltage regulator since all the other modes are strictly worse (see tables 1, 3). Also since VLPR and VLPW modes have the same characteristics as 2MHz Run and Wait modes respectively (tables 1, 5), we do not need to treat VLPW and VLPR as separate modes but instead we can think them as equivalent to 2MHz Run and Wait. Since there is only one sleep mode we need to consider in the simulations (LLS without USB voltage regulator) it will be referred simply as the sleep mode in this chapter.

The run modes and their respective idle modes and bus characteristics used are from tables 5, 7. We assume that during the wake-up sequence the current consumption is the same as in the run mode that follows.

We also assume that the transition from the sleep mode into any run mode takes an equal amount of time regardless of which run mode was used before entering the sleep mode. This might not always be the case because changing the core frequency might require some extra operations on top of the ones needed for the sleep. The time it takes to transition directly from one run mode to another is 0.3ms based on the transition of $VLPR \rightarrow Run$ (table 3). However, we rarely see this in the simulations because the transition from one core frequency to another usually has a sleep sequence in between.

For these simulations we assume that the time it takes to enter a sleep mode is 0ms. We use this because the time to enter a sleep mode is usually short [15].

We use the algorithms we present in section 4 to simulate plausible example scenarios. We make some small modifications to the algorithms to be able gather more detailed power consumption profiles of the examples. The modifications are essentially just storing intermediate values from the algorithm so that we can plot the power consumption by time.

5.1 Simple System Example

Consider a simple message scheme with devices d_1 and d_2 with $T_1 = 4\text{ms}, l_1 = 10\text{bytes}, P_{d_1,EXT} = 0\text{mA}$ and $T_2 = 8\text{ms}, l_2 = 10\text{bytes}, P_{d_2,EXT} = 0\text{mA}$. In a configuration this simple it is easy to see directly that the transmissions happen at $t = 0\text{ms}, 0\text{ms}, 4\text{ms}$

The current consumptions for this example with all the different power managers can be seen in table 10. There are only 3 distinct schemes in terms of power modes for this example so we only need to compare those 3. In table 11 the power modes used for each part of the transmission in all the distinct cases are shown. The difference between the one frequency with no sleep and sleep is very simple. The two sleep modes used during inactive periods causes the difference in current consumption. Note that the transition between the first two transitions is idle also in the sleep

Table 10: Current consumption of example 1 on different power managers.

Power manager	Average current consumption
One core frequency, no sleep	6.41mA
One core frequency, always sleep	2.27mA
One core frequency, sender based sleep	2.27mA
One core frequency, predictive sleep	2.27mA
Optimal, interrupt based wake-up	1.89mA
Optimal, preemptive wake-up	1.89mA

version because the transmission of the second message starts immediately after finishing the previous message. In the optimal scheme a sleep mode is used between the two overlapping transmissions. In that transition sleep mode is entered and a wake-up sequence starts immediately. This is because in the model we use the current used is lower for entering a sleep mode and then waking in a lower core frequency mode even though the direct core frequency change is faster. The delay itself does not result in an increase of current consumption due to the fact that the external current consumption is zero for all the external devices. However, it is unlikely that the direct clock speed change on a real hardware is less efficient than the transfer through a sleep mode and instead this is an oversight in the model.

Table 11: Power modes used for example 1 on different power managers.

Step	One frequency, no sleep	One frequency, sleep	Optimal
Transmission t = 0ms	8MHz	8MHz	8MHz
Transition	Idle	Idle	Sleep
Transmission t = 0ms	8MHz	8MHz	2MHz
Transition	Idle	Sleep	Idle
Transmission t = 4ms	8MHz	8MHz	2MHz
Transition	Idle	Sleep	Sleep

Using the 2 MHz mode would be more efficient but that mode is too slow to process all the transmissions in time. If the transmissions were allowed to surpass the cycle length (8 in this case) using the 2 MHz core frequency mode would be optimal.

For more complicated transmission schedules the list of power modes used for each transmission becomes very long and thus we omit that table from the following examples.

5.2 Dense System Example

Consider a dense example message scheme with devices d_1 and d_2 with $T_1 = 4\text{ms}, l_1 = 10\text{bytes}, P_{d_1,EXT} = 10\text{mA}$ and $T_2 = 5\text{ms}, l_2 = 10\text{bytes}, P_{d_2,EXT} = 10\text{mA}$. The hyperperiod we need to investigate is of $t = lcm(T_1, T_2)$ long which is $t = lcm(4, 5) =$

20ms. Using these values we generate the list of transmissions and because the length of the message and the external power consumption are the same the transmissions by different devices do not need to be separated. The times when transmission happen are at $t = 0\text{ms}, 0\text{ms}, 4\text{ms}, 5\text{ms}, 8\text{ms}, 10\text{ms}, 12\text{ms}, 15\text{ms}, 16\text{ms}$.

Table 12: Current consumption of example 2 on different power managers.

Power manager	Average current consumption
One core frequency, no sleep	9.87mA
One core frequency, always sleep	7.69mA
One core frequency, sender based sleep	7.69mA
One core frequency, predictive sleep	7.59mA
Optimal, interrupt based wake-up	7.39mA
Optimal, preemptive wake-up	6.52mA

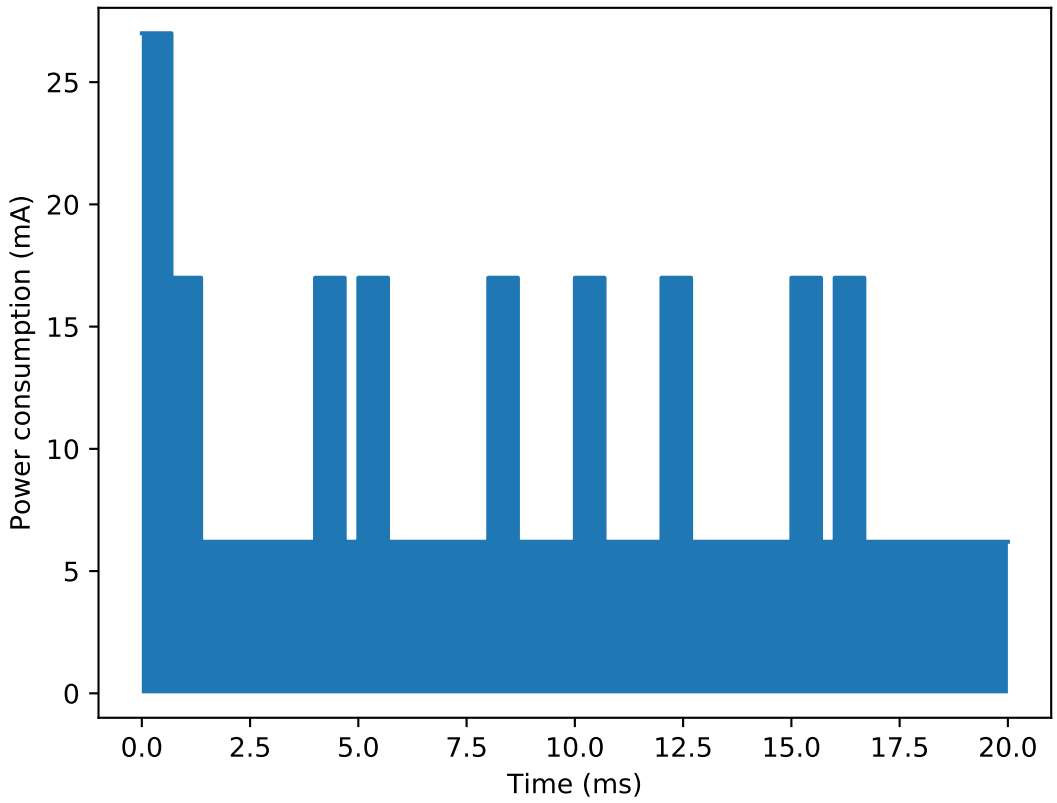

Figure 18: The power consumption plot of the no sleep scheme.

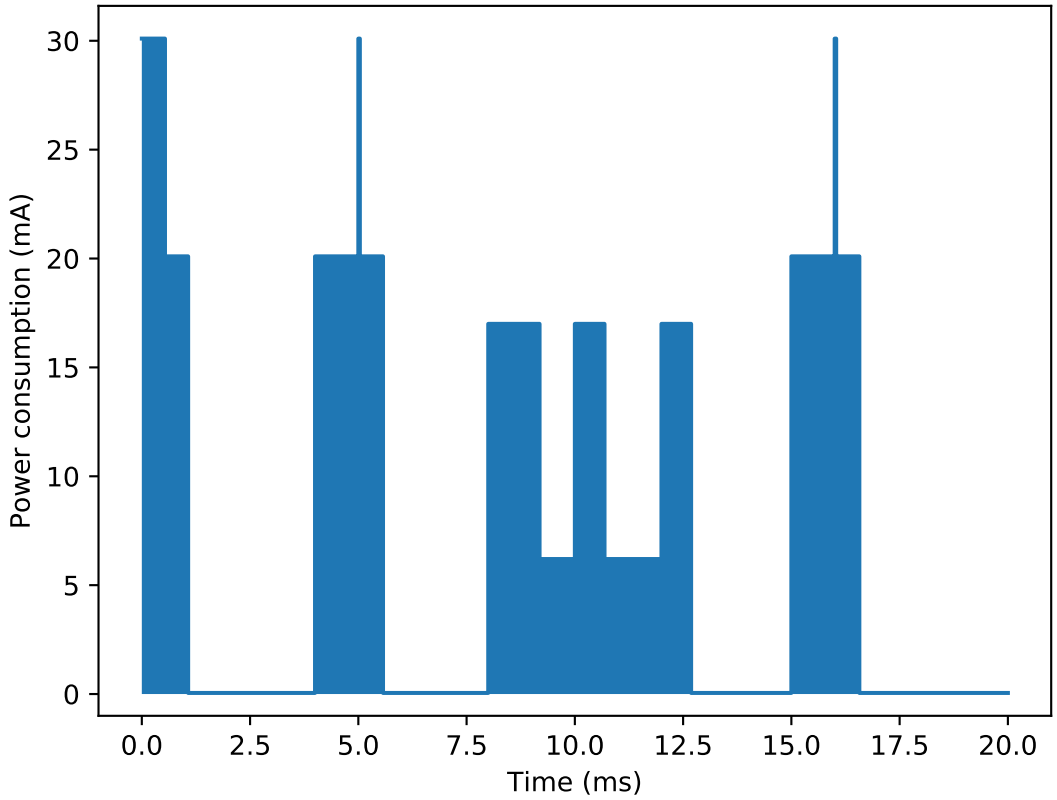

Figure 19: The power consumption plot of the optimal power manager with interrupts.

The current consumptions with different power management schemes for the message scheme is in table 12. The current consumption plots for the power management schemes of one core frequency with no sleep, optimal with interrupt based wake-ups, and optimal with preemptive wake-ups are in figures 18, 19, and 20.

Staying in idle mode is considerably more expensive than staying in a sleep mode and in setups where there is a considerable amount of inactive time this increases the current consumption considerably. Like in this example the no sleep scheme has a considerably higher current consumption than any other scheme as can be seen in table 12.

The differences between one core frequency schemes with different sleep schemes are small because most transmissions are either overlapping or far enough apart. This is because the only place where current could be saved is by going into idle mode instead of sleep. The maximum amount of current saved for each inactive section is $T_{WAKEUP} \cdot P_{EXT}$. This happens when the inactive approaches zero without reaching it. The device would enter a sleep mode but would need to wake up immediately after initiating the sleep transition. So for this example the decrease in average current consumption for one sleep-idle swap is at most $0.5\text{ms} \cdot 10\text{mA}/20\text{ms} = 0.25\text{mA}$. There

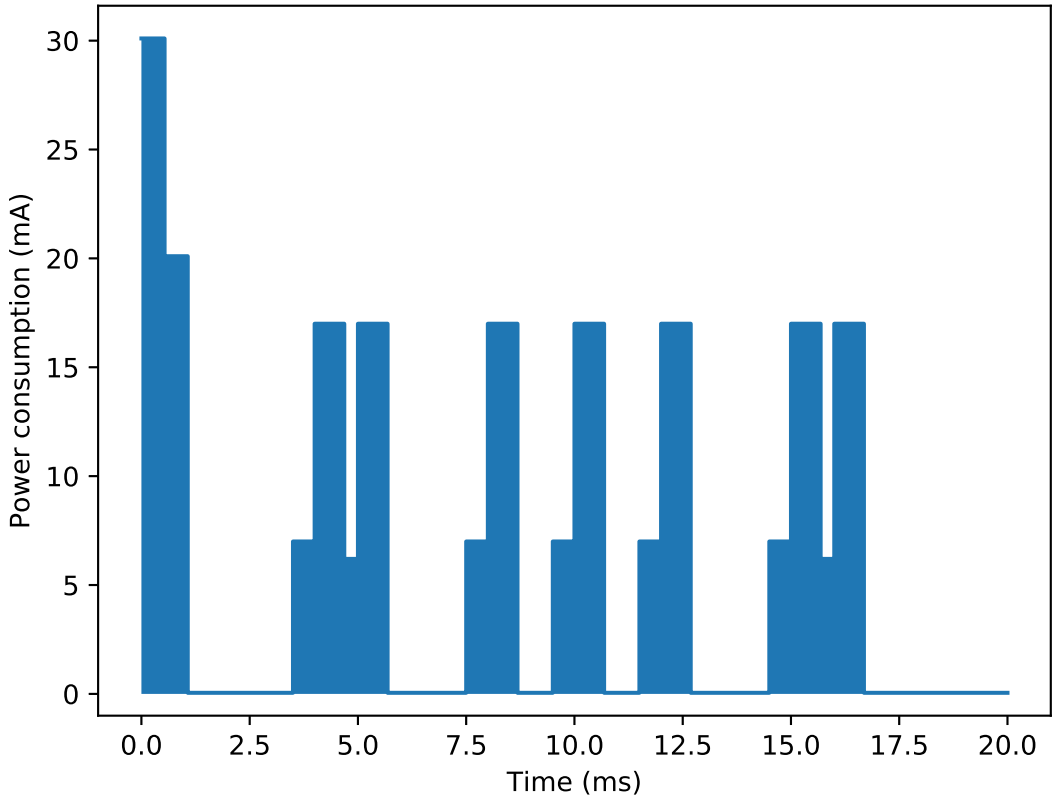

Figure 20: The power consumption plot of the optimal power manager with preemptive wake-ups.

are only a few of these so the difference ends up being 0.1mA for this example. In a more dense transmission setup, where delaying one transmission may delay the next transmission, the possible current saving might be higher.

Even the interrupt based optimal scheme saves little. Most of the savings over the one core frequency with predictive sleep power manager comes from changing core frequency and as the external current consumptions do not vary across different transmissions the difference in current consumption ends up being modest. From figure 19 we can see that the transmissions at 8ms, 10ms, and 12ms are processed at a different core frequency from the rest.

The preemptive wake-ups saves a lot as it basically reduces the time the external device needs to be active by the wake-up time of the receiver every time the receiver is in a sleep mode. The reason can be seen clearly in figures 19 and 20. The high plateaus are much shorter in duration in figure 20.

5.3 Sparse System Example

Consider a plausible real world example with 4 external devices with intervals of 1 second, 2 seconds, 5 seconds, and 10 seconds with the characteristics stated in table 13. Calculating the average current consumption with the different schemes gives the values in table 14. For sparse message schedules like this the maximum possible current savings over the scheme with always sleep is modest. In this case the current savings is only $\frac{0.128\text{mA}-0.120\text{mA}}{0.128\text{mA}} \cdot 100\% = 6.25\%$. For an interrupt based wake-up scheme the saving is even smaller. This shows the importance of using sleep modes as the simplest power manager with a sleep mode has a tenfold improvement in power consumption over a power manager with no sleep.

Table 13: Message characteristics of example 3.

Device	Period	Message length	External current consumption
Device 1	1000ms	10 bytes	1mA
Device 2	2000ms	1 bytes	10mA
Device 3	5000ms	10 bytes	10mA
Device 4	10000ms	100 bytes	100mA

Table 14: Current consumption of example 3 on different power managers.

Power manager	Average current consumption
One core frequency, no sleep	1.66mA
One core frequency, always sleep	0.128mA
One core frequency, sender based sleep	0.128mA
One core frequency, predictive sleep	0.128mA
Optimal, interrupt based wake-up	0.123mA
Optimal, preemptive wake-up	0.120mA

6 Discussion

In the measurements we do, the power consumption does not exactly match the theoretical model as is seen in figure 5. The graph should be linear as seen in function 10. Instead it is slightly curvy; in fact even the values in the data sheet don't result in a linear plot. Also when taking the bus speed into account some of the core frequencies are not optimal for any transmission. This is seen from figure 7. Every core frequency should be less efficient than the previous if they followed the theoretical model. Even so the processor we use in the measurements roughly follow the well-established theory.

Deciding on which sleep mode to enter based on the previous transmission did not save any power on any of the examples in the simulations. Although in these examples they did not it is possible for it to save power. However, it seems these cases are rare and it is intuitively understandable. The type of the previous transmission has little impact on the following transmission. The only thing it ensures is that the next transmission is less likely to be the same type as the previous one.

The predictive sleeping schedule saves very little power. The wake-up time from sleep is very short for the device we use in the measurements. In order to save power with the predictive sleeping schedule the idle periods between two messages has to be about as short as the wake-up time or shorter. Due to this worthwhile idle periods are rare. However, it is possible that for a device with different characteristics the predictive sleeping schedule would save power more frequently.

Even the optimal sleeping with interrupt wake-ups saves only a modest amount of power, only around 10% at most. The preemptive wake-up also saves modestly. However, due to the bus speeds and bus efficiencies with the system we use, large power savings are not likely. Consider a transmission schedule where the receiver always runs at a constant 16 MHz clock speed. For messages where $P_{EXT} = 0$ and the message is long so the wake-up period is not relevant, the power saving is $\frac{Beff_{MAX} - Beff_{16MHz}}{Beff_{MAX}} = \frac{2790-2520}{2790} \approx 11\%$. Where $Beff_m$ is the bus efficiency in mode m. For messages where P_{EXT} is large so the receiver power consumption does not matter the power saving is $\frac{Bv_{MAX} - Bv_{16MHz}}{Bv_{MAX}} = \frac{32800-25500}{32800} \approx 22\%$. Where Bv_m is the bus speed in mode m. This means for a setup that uses all core frequencies can save at most 22% over a system which only uses 16 MHz core frequency. In these calculations we assume that the power consumption during sleep is negligible.

The calculation model might result in inaccurate values in some cases. In the periodic message scheme the transition from the last message into the first transmission is not calculated correctly. In the model we use at time $t = 0$ the processor is allowed to be in a different mode than at the end of the hyperperiod. This means the power manager would not be able to follow the calculated transition during this moment. However, in most cases the difference is small because the transmission period is usually long and the one erroneously calculated transition is only a small portion of the total power consumption. Fixing this problem is not trivial because we cannot know in advance what mode the processor should start in and if the last messages of the previous period overflow to the next cycle. One possible solution to this is to calculate many hyperperiods until the starting and ending mode

in the hyperperiod match. This can take many iteration rounds because a change at the start of the hyperperiod might propagate to the end of the hyperperiod.

Another unrealistic result in the calculations is that the power consumption of the same external device is calculated multiple times if a transmission is delayed so much that it overflows past the next transmission. Although it might not be a bad way to model a system because delaying a transmission over the next transmission could cause some other unwanted consequences and giving it a penalty could be desirable.

For practical purposes in the example systems the simple power manager of always entering sleep mode is probably the most appropriate. The power saving with the more advanced power managers is modest and the improvement in power consumption comes with a cost of increased complexity. Also real applications would most likely have some uncertainty in the timings which would make the schedule of processor modes inaccurate and thus less effective.

With different system characteristics better power savings are possible and perhaps for the correct type of system it is worth the increase in complexity. This work only explores one type of processor and with a capped bus speed. A different processor or an unlimited bus speed would provide different results. This is something that could be explored further.

The model we use considers the transmissions as the only activity in the system. Usually, however, systems also have some other processing tasks that need to be completed. This could for example be analysis done with the received messages. This could be taken into consideration in the power manager. It would bring considerable complexity to the power manager but it would be able to model many real life systems more accurately.

Although the model and the algorithms would support periodic transmission schedules with offsets or even arbitrary transmissions schedules with minor alterations to the algorithms, these are not explored in this work. Certain offsets or arbitrary transmission schedules might benefit significantly from the proposed power management schemes. This can especially be the case if two messages have the same period but with a different offset.

7 Conclusions

This thesis aims to find the best power manager for a system where a single device is connected to multiple external devices through a shared blocking communication line. We allow the device to transition between different run and sleep modes whereas the external devices stay in an active mode when transmitting or ready to transmit and sleep otherwise. With this we aim to minimize the power consumption of the entire system.

We perform measurements with a Teensy 3.2 microcontroller development board that has an ARM Cortex-M4 microcontroller. We measure the power consumption and the communication bus speed in different run and sleep modes and compare these to the existing models of power consumption. We also measure the transition times between different power modes. We use these measured values to analyze the different power managers we propose.

We present six different power managers. The two simplest power managers are simply using a single run mode across the entire task cycle, one of them uses a sleep mode and the other uses an idle mode whenever inactive. The third power manager we present uses a single run mode and enters a sleep mode based on which device sent the previous message. The fourth power manager enters a sleep mode based on how long it takes until the next transmission arrives. This power manager also uses only a single run mode. The final two run modes use a precalculated schedule to determine which run mode to use and when to enter a sleep mode. The difference between the two power managers is that one wakes up on its own and the other wakes up to an interrupt when the message transmission starts. We provide algorithms for calculating the transition times for each of the power algorithms for a given message schedule.

We use the measured values we gathered to simulate various example systems. Using these simulations we manage to calculate the power consumption for the example systems with the different power managers we propose. We find that the power savings with the different power managers for the examples vary from modest to none depending on the transmission system and the power manager. The optimal preemptive power manager is the most efficient out of all the power managers we test. It depicts a power saving of about 10% over a power manager which always sleeps and uses only one run mode. The interrupt based optimal power schedule also saves a noticeable amount, about 5% on average. The other power managers save barely anything. The power manager that is not allowed to use sleep modes naturally consumes significantly more power in system with inactive sections. Using a sleep mode at all can improve the power consumption significantly, a tenfold or higher increase in efficiency is possible in some systems.

References

[1] Enhanced I2C library for Teensy 3.x & LC devices. https://github.com/nox771/i2c_t3 Accessed: 2019-04-09

[2] I^{2}C-bus specification and user manual. https://www.nxp.com/docs/en/user-guide/UM10204.pdf Accessed: 2019-04-09

[3] K20 Sub-Family Data Sheet. https://www.nxp.com/docs/en/data-sheet/K20P64M72SF1.pdf. Accessed: 2019-04-09

[4] K20 Sub-Family Reference Manual. https://www.pjrc.com/teensy/K20P64M72SF1RM.pdf. Accessed: 2019-04-09

[5] Low power library for the Teensy LC/3.2/3.5/3.6 class microcontrollers. https://github.com/duff2013/Snooze Accessed: 2019-04-09

[6] sleep.h https://github.com/duff2013/Snooze/blob/master/utility/sleep.h Accessed: 2019-05-09

[7] Teensy 3.2 Schematic. https://www.pjrc.com/teensy/schematic.html. Accessed: 2019-04-09

[8] Teensy USB Development Board. https://www.pjrc.com/teensy/ Accessed: 2019-05-23

[9] Wire Library. https://www.pjrc.com/teensy/td_libs_Wire.html. Accessed: 2019-04-09

[10] https://commons.wikimedia.org/wiki/File:PIC18F8720.jpg Accessed: 2019-06-17

[11] Amdahl, G.M. 1967, Validity of the single processor approach to achieving large scale computing capabilities, *AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference*, AFIPS 1967, pp. 483-485. doi:10.1145/1465482.1465560

[12] Aydin, H., Devadas, V., Zhu, D. 2006, System-level energy management for periodic real-time tasks, *Proceedings - Real-Time Systems Symposium*, pp. 313-322. doi: 10.1109/RTSS.2006.48

[13] Aydin, H., Melhem, R., Mossé, D., Mejía-Alvarez, P. 2001, Determining optimal processor speeds for periodic real-time tasks with different power characteristics, *Proceedings - Euromicro Conference on Real-Time Systems*, pp. 225-232. doi:10.1109/EMRTS.2001.934038

[14] Bambagini, M., Marinoni, M., Aydin, H., Buttazzo, G. 2016, Energy-aware scheduling for real-time systems: A survey, *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 1. doi:10.1145/2808231

[15] Benini, L., Bogliolo, A., De Micheli, G. 2000, A survey of design techniques for system-level dynamic power management, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299-316. doi:10.1109/92.845896

[16] Cardoso, J., Coutinho, J.G., Diniz, P. 2017, Embedded Computing for High Performance. 1st edition. Morgan Kaufmann

[17] Disser, Y., Müller-Hannemann, M., Schnee, M. 2008, Multi-criteria shortest paths in time-dependent train networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5038 LNCS, pp. 347-361. doi:10.1007/978-3-540-68552-4_26

[18] Lee, Y.-H., Reddy, K.P., Krishna, C.M. 2003, Scheduling techniques for reducing leakage power in hard real-time systems, *Proceedings - Euromicro Conference on Real-Time Systems*, pp. 105-112. doi:10.1109/EMRTS.2003.1212733

[19] Ling, S., Sanny, J., Moebs, W. 2016, University Physics Volume 1. https://openstax.org/details/books/university-physics-volume-1 Accessed: 2019-06-17

[20] Ling, S., Sanny, J., Moebs, W. 2016, University Physics Volume 2. https://openstax.org/details/books/university-physics-volume-2 Accessed: 2019-05-16

[21] Liu, C.L., Layland, J.W. 1973, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46-61. doi:10.1145/321738.321743

[22] Lu, Y.-H., Chung, E.-Y., Šimunić, T., Benini, L., De Micheli, G. 2000, Quantitative comparison of power management algorithms, *Proceedings -Design, Automation and Test in Europe*, pp. 20-26. doi:10.1109/DATE.2000.840010

[23] Luo, J., Jha, N.K. 2000, Power-conscious joint scheduling of periodic task graphs and aperiodic tasks distributed real-time embedded systems, *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pp. 357-364. doi:10.1109/ICCAD.2000.896498

[24] Martin, T.L., Siewiorek, D.P. 2001, Nonideal battery and main memory effects on CPU speed-setting for low power, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 29-34. doi:10.1109/92.920816

[25] Martin, S. M, Flautner, K., Mudge, T., Blaauw, D. 2002, Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads, *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*, pp. 721-725. doi:10.1145/774572.774678

[26] Mittal, S. 2014, A Survey of Techniques For Improving Energy Efficiency in Embedded Computing Systems, *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 4, pp. 440-459. doi:10.1504/IJCAET.2014.065419

[27] Rusu, C., Melhem, R., Mossé, D. 2003, Maximizing Rewards for Real-Time Applications with Energy Constraints, *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 4, pp. 537-559. doi:10.1145/950162.950166

[28] Srivastava, M.B., Chandrakasan, A.P., Brodersen, R.W. 1996, Predictive system shutdown and other architectural techniques for energy efficient programmable computation, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 42-55. doi:10.1109/92.486080

[29] Veendrick, H. J. M. 1984, Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits, *IEEE Journal of Solid-State Circuits*, vol. 19, no. 4, pp. 468-473. doi:10.1109/JSSC.1984.1052168

[30] Yiu, J. 2015. The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors, 2nd Edition. 2nd edition. Newnes.

[31] Wolf, W.H. 1994, Hardware–Software Co–Design of Embedded Systems, *Proceedings of the IEEE*, vol. 82, no. 7, pp. 967-989. doi:10.1109/5.293155

[32] Zhu, D., Aydin, H. 2009, Reliability-aware energy management for periodic real-time tasks, *IEEE Transactions on Computers*, vol. 58, no. 10, pp. 1382-1397. doi:10.1109/TC.2009.56