Voltage and frequency scaling in an embedded microprocessor to enable the implementation of dynamic voltage and frequency scheduling for power management.

Master of Science Thesis

Jonas Höglund

December 13, 2009

Supervisors:
Magnus Persson, KTH
Barbro Claesson, ENEA
Detlef Scholle, ENEA

Examiner:
Martin Törngren, KTH
Examensarbete MMK 2009:98 MDA 358

Frekvens och spänningskalning i en inbyggd mikroprocessor; för att möjliggöra implementationen av schemaläggning med stöd för dynamisk frekvens- och spänningsreglering för energihantering.

Jonas Höglund

Sammanfattning

Att minimera energiförbrukning är en mycket viktig del av många utvecklingsprojekt av inbyggda system. Utmaningen består av att minimera systemets energiförbrukning samtidigt som det förväntas leverera en kvalitativ användarupplevelse. Den huvudsakliga metoden som används för att minimera energiförbrukningen en mikroprocessor under programexekvering är att reducera dess klockfrekvens och matningsspänning.

För att möjliggöra implementationen av reallidsschemaläggning med stöd för dynamisk justering av processorns spänning och frekvens genomförs en utförlig studie av en Freescale i.MX31-mikroprocessor. Undersökningen fokuserar på relationen mellan energiförbrukning och spännings- och frekvensinställningar. För att möjliggöra frekvens- och spänningskalning i reaktidssammanhang utförs dessutom en tidsanalys av föreslagna spännings- och frekvensjusteringsmetoder och deras lämplighet för användning i reaktidssystem utvärderas.

Abstract

Minimizing power consumption is a critical part of many embedded design projects. The challenge is to limit the power consumption of the system and at the same time provide satisfactory service to the user. With regards to microprocessors, the main way of limiting power consumption during program execution is to adjust the frequency and the voltage at which the processor operates.

To enable the implementation of real-time scheduling with support for dynamic voltage and frequency scaling, an exhaustive study of the Freescale i.MX31 embedded microprocessor is performed. Specifically, the power consumption relationship to all relevant frequency and voltage settings is investigated. In addition, to enable frequency and voltage scaling in a real-time environment, a complete timing analysis of proposed voltage and frequency scaling methods is performed and their suitability for use in real-time systems is evaluated.

To integrate voltage and frequency scaling capability into OSE RTOS, a software module has been developed for this operating system. The module enables OSE processes to request voltage and frequency changes using a signaling interface. The module has been tested for robustness and its correct operation together with a real-time dynamic voltage frequency scheduler developed at Enea has been verified. Finally, an evaluation of the power consumption of a test program running on the integrated system is performed.
## Contents

Contents................................................................................................................................. 4  
List of figures .................................................................................................................................. 5  
1 Introduction.................................................................................................................................. 6  
   1.1 Background .............................................................................................................................. 6  
   1.2 Problem Statement .................................................................................................................. 6  
   1.3 Method .................................................................................................................................... 7  
   1.4 Delimitations .......................................................................................................................... 7  
2 Power management in mobile devices...................................................................................... 8  
   2.1 Battery technology and the need for power efficiency ........................................................... 8  
   2.2 The CPU as a consumer of energy .......................................................................................... 8  
   2.3 The computational needs of mobile handhelds ....................................................................... 9  
   2.4 Summary ............................................................................................................................... 10  
3 Addressing timing requirements of computing systems .......................................................... 11  
   3.1 Real-time systems .................................................................................................................. 11  
   3.2 Worst case execution time ...................................................................................................... 14  
   3.3 Summary ................................................................................................................................ 15  
4 Power management techniques for real-time systems ............................................................. 16  
   4.1 Sources of power consumption ............................................................................................. 16  
   4.2 Dynamic Voltage Frequency Scaling .................................................................................... 16  
   4.3 Summary ................................................................................................................................ 17  
5 Hardware support for power management ............................................................................... 18  
   5.1 i.MX31 support for frequency scaling .................................................................................... 19  
   5.2 i.MX31 support for automatic DVFS ..................................................................................... 22  
   5.3 MC13783 support for voltage scaling .................................................................................... 24  
   5.4 i.MX31 Frequency/Voltage operating ranges ....................................................................... 25  
   5.5 Summary ................................................................................................................................ 26  
6 Supporting software platforms .................................................................................................. 27  
   6.1 U-boot 1.3.3 ............................................................................................................................. 27  
   6.2 OSE 5.4 .................................................................................................................................... 27  
   6.3 OSE Bios interface .................................................................................................................. 28  
   6.4 Summary ................................................................................................................................ 29  
7 Design description ...................................................................................................................... 30  
   7.1 Voltage scaling ....................................................................................................................... 30  
   7.2 Frequency scaling ................................................................................................................... 31  
8 Results.......................................................................................................................................... 33  
   8.1 Test cases ............................................................................................................................... 33  
9 Discussion..................................................................................................................................... 36  
   9.1 Determination of CPU power consumption and timing characteristics ......................... 36  
   9.2 Verification of OSE DVFS module ......................................................................................... 42  
10 Future Work ............................................................................................................................... 44  
   10.1 Summary of results ............................................................................................................... 44  
   10.2 Requirements ....................................................................................................................... 45  
   10.3 Future Work ......................................................................................................................... 46  
11 Appendix A.................................................................................................................................. 47  
   11.1 Test 1: CPU 100% load & Test 2: CPU 0% load ................................................................. 47  
   11.2 Test 3: PLL vs. Post divider scaling ..................................................................................... 47  
   11.3 Test 5: Timing characteristics of frequency scaling ........................................................... 47  
   11.4 Test 7: Power consumption of integrated system ............................................................... 48  
12 Bibliography................................................................................................................................ 49
List of figures

Figure 1 Performance/ Stamina Gap, SMRTSPDMFWP Rev 1, 04/2009, Freescale Semiconductor 8
Figure 2 Real-Time Task.................................................................11
Figure 3 Schedulability test for rate monotonic scheduling.................................13
Figure 4 Schedulability test for earliest deadline first........................................13
Figure 5 DVFS operation, eXtreme Energy Conservation, 02/2008, Freescale Semiconductor ....16
Figure 6 i.MX31 ADS, i.MX31ADS Application Development System User's Manual, 03/2006, Freescale Semiconductor.................................18
Figure 7 Overview of the registers involved in frequency scaling..........................19
Figure 8 System clock generation, MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor........20
Figure 9 MCU PLL Control Register, MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor...20
Figure 10 MCU PLL Output frequency (Fvco), MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor.................................................................21
Figure 11 PDR0 Post divider register 0, MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor, .................................................................21
Figure 12 Power management control register (PMCR0), MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor.................................21
Figure 13 Frequency update procedure ...................................................22
Figure 14 i.MX31 automatic DVFS approaches, IMX31POWERWP Rev. 1, 12/2006, Freescale Semiconductor.................................................................23
Figure 15 DVFS hardware mechanism, IMX31POWERWP Rev. 1, 12/2006, Freescale Semiconductor.................................................................23
Figure 16 MC13783 SPI Interface, MC13783GPLDRM Rev. 1.1, 4/2008, Freescale Semiconductor. .................................................................24
Figure 17 MC13783 Register 24, MC13783GPLDRM Rev. 1.1, 4/2008, Freescale Semiconductor...24
Figure 18 Supported voltage levels SW1A, MC13783GPLDRM Rev. 1.1, 4/2008, Freescale Semiconductor .................................................................24
Figure 19 Allowed voltage levels, MCIMX31_5 Rev. 4.2, 11/2008, Freescale Semiconductor........25
Figure 20 Allowed PLL values, MCIMX31_5 Rev. 4.2, 11/2008, Freescale Semiconductor........25
Figure 21 PLL switch times, MCIMX31_5 Rev. 4.2, 11/2008, Freescale Semiconductor. .................................................................26
Figure 22 Current Consumption OSE/U-boot i.MX31 ADS........................................36
Figure 23 Power Consumption OSE/U-boot i.MX31 ADS ........................................37
Figure 24 Switching to and from 100 - 399 MHz...............................................38
Figure 25 Switching to and from 100 - 200 MHz...............................................38
Figure 26 Switching to and from 200 - 399 MHz...............................................38
Figure 27 Switch from 1,20 V to 1,40 V..........................................................41
Figure 28 Switch from 1,40 V to 1,20 V..........................................................41
Figure 29 Switch from 1,40 V to 1,60 V..........................................................41
Figure 30 Switch from 1,60 V to 1,40 V..........................................................41
Figure 31 Current consumption integrated DVFS system.................................42
Figure 32 Power consumption integrated DVFS system.................................43
Figure 33 All 532, 1,60 V.................................................................................48
Figure 34 All 532, 1,40 V.................................................................................48
Figure 35 All 399, 1,40 V.................................................................................48
Figure 36 All 399, 1,20 V.................................................................................48
Figure 37 Idle 100 MHz, 1.20 V.........................................................................48
Figure 38 Dynamic P3, 1.20 V.........................................................................48
1 Introduction

1.1 Background

During the ENEA project DySCAS\(^1\), an intelligent automotive networked middleware called SHAPE\(^2\) was developed. The GEODES\(^3\) project is a continuation of this work which aims to integrate Quality of Service (QoS) functionality with a focus on power consumption in the SHAPE middleware. Preliminary work along these lines has been performed during earlier master thesis projects. This master thesis aims to continue this work with a focus on CPU power consumption.

1.2 Problem Statement

Minimizing power consumption is a critical part of many embedded design projects. The challenge is to limit the power consumption of the system while still providing satisfactory services to the user. With regards to the microprocessors, the main way of limiting power consumption during program execution is to adjust the frequency and the voltage at which the processor operates.

1.2.1 Determination of CPU power consumption and timing characteristics

To reduce the power consumption of a microprocessor executing instructions, the processor frequency and supply voltage can be lowered. Preliminary studies have been performed on the target i.MX31 processor and a non-linear relationship between processor frequency and power consumption has been determined to exist. An exhaustive study of all possible frequency and supply voltage settings will be performed. How does the power consumed by the processor depend on the frequency and voltage settings? For use in real-time systems, deterministic timing characteristics must be determined for all state transitions involved in power management. What are the timing characteristics of the proposed frequency and voltage scaling methods? The type of power management that can be implemented is inherently limited by the specific hardware architectural support on the target processor. What hardware functionality does the i.MX31 ADS have to support voltage and frequency scaling and what is its suitability of use in a real-time system?

1.2.2 Development of driver functionality and interface to enable implementation of Dynamic Voltage Frequency Scaling (DVFS) scheduler

To enable the implementation of a real-time DVFS scheduling algorithm a device driver controlling access to voltage and frequency scaling hardware will be developed. The implementation of power management on the processor must therefore take real-time issues into consideration. What is required of the dvfs driver to enable the implementation of power management in a real-time system? The overall goal of this thesis work is to develop the capability to change the

\(^1\) DySCAS IST-034904 – Dynamically Self-Configuring Automotive System

\(^2\) SHAPE – Self-configuring High-Availability Policy-based system for Embedded systems

\(^3\) GEODES Itea 2 – 07013 – Global Energy Optimization for Distributed Embedded Systems.
frequency and the voltage of the CPU dynamically during the execution of real-time tasks. scheduler into the OSE™ operating system. How can the DVFS scheduler be implemented in OSE?

1.3 Method

The master thesis project will start with a study of books, research papers, technical manuals as well as earlier master thesis work relating to voltage and frequency scaling in an embedded CPU and power saving methods for embedded systems in general. Test procedures developed by earlier master thesis students will be the starting point of performing an exhaustive power consumption study. Once completed, a design description will be written based on the results of the study. Finally, the suggested design will be implemented and tested.

1.4 Delimitations

The time limit for the thesis project is 20 weeks. A literature study, a design description, an implementation, a presentation and a written report has to be completed during this period. The hardware platform will be the Freescale i.MX31 ADS development board with an i.MX31 ARM11 CPU. The software platforms utilized will be the U-boot bootloader and OSE 5.4.

OSE – Enea Embedded Operating System
2 Power management in mobile devices

This chapter presents a specific user case in the form of a generic mobile device to give a context for the implementation of power management techniques. Goals that should be attained by the system are presented in the form of software design requirements (SWDRQM).

2.1 Battery technology and the need for power efficiency

Of all consumer products, hand held devices such as mobile phones and PDA:s place a very high premium on small size and weight. Since the battery is already the largest and heaviest component of a phone, increased power must come from increasing the energy efficiency of existing batteries, not by increasing their size and weight. Early phones used lead-acid batteries which made them large and bulky. Technological progress has made batteries smaller and more energy efficient. Modern mobile phones use lithium ion batteries. Figure 1 illustrates the fact that although lithium ion technology has progressed steadily forward it has not kept up with increasing energy demands for mobile platforms such as multimedia mobile phones and PDA:s with large color displays. While technologies such as fuel cells seem to have great potential in regards to increasing energy efficiency, it is not expected that they will eliminate the need for power efficiency due to the ever increasing demands of power hungry applications.

Based on this discussion, the following design criteria should be observed when designing mobile devices:

SWREQ1: The system should be designed in a manner that minimizes the energy it consumes.

Figure 1 Performance/ Stamina Gap, SMRTSPDMFWP Rev 1, 04/2009, Freescale Semiconductor

2.2 The CPU as a consumer of energy

In the context of overall system energy efficiency, the power consumption of the CPU should be given due attention since it has the capacity to consumes a considerable amount of energy.
SWREQ2: The CPU should perform its required computations in the most energy-efficient manner possible while observing all other requirements.

However, it is not desirable to introduce a large power management overhead into the system. To avoid such overhead, dedicated power management hardware circuitry should be utilized whenever possible to relieve the CPU of unnecessary work.

SWREQ3: The CPU should utilize all appropriate on-chip hardware resources for power management to reduce power management overhead.

### 2.3 The computational needs of mobile handhelds

The computational needs of hand-held devices can be summed up into two parts, real-time and non real-time. Real-time computing systems must react dynamically to the state changes of an environment, whose evolution depends on human behavior, a natural or artificial phenomenon or an industrial plant. Real-time applications span a large spectrum of activities; examples include production automation, embedded systems, telecommunication systems, automotive applications, nuclear plant supervision, scientific experiments, robotics, multimedia audio and video transport and conditioning, surgical operation monitoring and banking transactions. In all these applications, time is the basic constraint to deal with and the main concern for appraising the quality of service provided by computing systems.

Application requirements lead to differentiation between hard and soft real-time constraints. Applications have hard real-time constraints when a single failure to meet timing constraints may result in an economic, human or ecological disaster. A time fault may result in a deadline being missed, a message arriving too late, an irregular sampling period, a large timing dispersion in a set of ‘simultaneous’ measurements, and so on. Soft real-time constraints are involved in those cases when timing faults cause damage whose cost is considered tolerable.

Real-time is a serious challenge for computing systems and its difficulties are often misunderstood. A real-time computing system must provide a time management facility; this is an important difference compared to conventional computing systems, since the value of data produced by a real-time application depends not only upon the correctness of the computation but also upon the time at which the data is available. An order which is computed right but sent late is a wrong command; it is a timing fault.

Real-time is one of the key requirements for cell phone designs. These phones require the ability to support both the base-band protocol (with strong real-time requirements) and end-user applications, including streaming media and other real-time applications. This is summed up in the following requirement:

SWREQ4: The system must meet the deadlines of all hard real-time tasks and strive to meet all soft deadlines.
2.4 Summary

- The need for power efficiency is unlikely to be relieved by new battery technology in the short term future. Power efficient computation should therefore be a prime concern in the design of new hand-held devices.
- To avoid introducing power management computational overhead, on-chip power management hardware should be used when appropriate.
- Some of the computations done in cellular devices have real-time constraints. This includes the ability to support base-band protocols (with strong real-time requirements) and end-user applications, including streaming media and other real-time applications.
- The problem of power efficient computation in mobile devices thus becomes: How can the power consumption be minimized under the constraint that all tasks must meet their deadlines?
3 Addressing timing requirements of computing systems

As described in the previous chapter, any power management implementation on a mobile device must take real-time issues into consideration. In this chapter, an overview of so-called real-time system concepts will be given. The purpose of this chapter is to present a framework for later discussions of power management under real-time constraints.

3.1 Real-time systems

Cottet et al describes a real time system as one in which time is the basic constraint to deal with and the main concern for appraising the quality of service. The value of data produced by such a system depends not only upon the correctness of the computation but also upon the time at which the data is available. An order which is computed right but sent late is a wrong command; it is a timing fault.1

In the context of real-time applications, the actions are called tasks and the organization of their execution by the processors of the computing architecture (sequencing, interleaving, overlapping, parallel computing) is called the real-time scheduling of tasks. The schedule must meet the timing constraints of the application; the procedure that rules the task execution ordering is called the scheduling policy.1

3.1.1 Basic concepts

Real-time tasks are the basic executable entities that are scheduled; they may be periodic or aperiodic, and have soft or hard real-time constraints. A task model can be described with the following main timing parameters.

- r, task release time, i.e. the triggering time of the task execution request.
- C, task worst-case computation time, when the processor is fully allocated to it.
- T, task period (valid only for periodic tasks).

![Figure 2 Real-Time Task](image)

The quality of scheduling depends on the exactness of these parameters, so their determination is an important aspect of real-time design. If the durations of operations like task switching, operating system calls, interrupt processing and scheduler execution cannot be neglected, the design analysis must estimate these durations and add them to the task computation times. That is why a deterministic behavior is required for the kernel, which should guarantee maximum values for these operations.
In a real-time system, tasks have timing constraints and their execution is bounded to a maximum delay that has to be respected. The objective of scheduling is to allow tasks to fulfill these timing constraints when the application runs in a nominal mode. A schedule must be predictable, i.e. it must be *a priori* proven that all the timing constraints are met in a nominal mode. When malfunctions occur in the controlled process, some alarm tasks may be triggered or some execution times may increase, overloading the application and giving rise to timing faults. In an overload situation, the objective of scheduling is to allow some tolerance, i.e. to allow the execution of the tasks that keep the process safe, although at a minimal level of service.

Scheduling a task set consists of planning the execution of task requests in order to meet the timing constraints:

- of all tasks when the system runs in the nominal mode;
- of at least the most important tasks (i.e. the tasks that are necessary to keep the controlled process secure), in an abnormal mode.

An abnormal mode may be caused by hardware faults or other unexpected events. In some applications, additional performance criteria are sought, such as minimizing the response time, reducing the jitter, balancing the processor load among several sites, limiting the communication cost, or minimizing the number of late tasks and messages and their cumulative lag.

The scheduling algorithm assigns tasks to the processor and provides an ordered list of tasks, called the planning sequence or the schedule. Off-line scheduling builds a complete planning sequence with all task set parameters. The schedule is known before task execution and can be implemented efficiently. However, this static approach is very rigid; it assumes that all parameters, including release times, are fixed and cannot adapt to environmental changes. Conversely, on-line scheduling, allows choosing at any time the next task to be elected. On-line scheduling therefore necessitates knowledge of the parameters of the currently triggered tasks. When a new event occurs the elected task may be changed without knowing in advance the time of this event occurrence. This better manages the unpredictable arrival of tasks at the price of a higher implementation overhead.

A scheduling algorithm results in a feasible schedule if all tasks meet their timing constraints. The determination of whether a periodic task set submitted to a given scheduling algorithm results in a feasible schedule is done by a schedulability test, specific to the scheduling algorithm. If the task set is periodic, it is sufficient to perform the schedulability test on the hyperperiod, the length of which is determined by the least common multiple of the periods of the tasks in the task set.

### 3.1.2 Scheduling policies

This section will cover two of the most utilized scheduling policies in systems with real-time constraints: rate-monotonic scheduling and earliest deadline first scheduling.

#### 3.1.2.1 Rate Monotonic Scheduling

Rate-monotonic scheduling (RMS) is a scheduling algorithm that can be used in real-time operating systems that build on static-priority scheduling. The static priorities are then assigned on the basis of the cycle duration of the job: the shorter the cycle duration is, the higher is the job's priority.
Liu & Layland (1973) proved that for a set of $n$ periodic tasks with unique periods, a feasible schedule that will always meet deadlines exists if the CPU utilization is below a specific bound (depending on the number of tasks). The schedulability test for rate monotonic scheduling is:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

**Figure 3** Schedulability test for rate monotonic scheduling

where $C_i$ is the computation time, and $T_i$ is the release period (with deadline one period later). For example $U \leq 0.82$ for $n = 2$. When the number of processes approaches infinity the value of this expression will approach the natural logarithm of 2.

A rough estimate is that RMS in the general case can meet all the deadlines if CPU utilization is 69%. The other 31% of the CPU time can be dedicated to lower-priority non real-time tasks. The rate monotonic scheduling algorithm has received widespread use in real-time systems mainly due to the ease of implementing it in real-time operating systems that builds on static priority scheduling. Another desirable trait that has contributed to its adoption is that under system overload, the highest priority tasks experience no performance degradation. This however comes at the expense of starvation of the lower priority tasks that are never allocated the CPU during overload conditions.

### 3.1.2.2 Earliest Deadline First

Earliest Deadline First (EDF) is a dynamic scheduling algorithm used in real-time operating systems (Liu and Layland, 1973). It places processes in a priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.

With scheduling periodic processes that have deadlines equal to their periods, EDF has a utilization bound of 100%. Thus, the schedulability test for EDF is:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,$$

**Figure 4** Schedulability test for earliest deadline first

That is, EDF can guarantee that all deadlines are met provided that the total CPU utilization is not more than 100%. So, compared to fixed priority scheduling techniques like rate-monotonic scheduling, EDF can guarantee all the deadlines of the system at a higher loading.

EDF has received widespread support in the academic community where it is viewed as superior to RMS due to the its higher schedulability bound, the ease of which it handles aperiodic tasks and the way in which it avoids unnecessary context switches (preemptions by higher priority tasks). However, EDF has not been widely adopted in industrial applications. One reason for this is that when the system is overloaded, the set of processes that will miss deadlines is largely unpredictable (it will be a function of the exact deadlines and time at which the overload occurs.) In addition most real-time operating systems implement some sort of static priority scheduling that is more suited to RMS. For research purposes however, patches exist that map deadlines to static priorities in such systems.
3.1.3 Impact of power management on scheduling algorithm

The prime concern when applying power management to a real-time system must be that the schedulability of the system is ensured. Concretely, this means integrating the delays introduced by transitioning to different power states into the schedulability condition for the utilized scheduling algorithm.

3.2 Worst case execution time

In addition to knowing the transition times between power states, to ensure schedulability, one must know the worst case execution times. This problem is not as straightforward as it could appear due to certain design solutions employed in all modern high-performance microprocessors. These solutions give rise to unpredictable timing behavior of the source code. Halang et. al. describes the situation in the following manner:

“In all hardware architecture design, an unfortunate situation can be observed: although technological advances offer immense possibilities for design and implementation of processor architecture suitable for embedded real-time applications, the processors produced are mainly designed with universal computing in mind. Except for simple microcontrollers with low performance and only basic features, modern processors exhibit serious drawbacks when employed in embedded real-time applications. The main reason for the poor suitability of current microprocessors for real-time computing is the mismatch of the global objectives in the design of universal processors and those dedicated to real-time applications.”

They further give examples of hardware solutions made with regard to universal computing that are contrary to the goal of predictability of execution time. These include pipelining, caching and virtual addressing. These measures try to optimize the behavior of processors and computers in the most common situations of computational processing. Embedded computer systems, however, operate in real-time mode. Most often, its requirements are demanding and it is of utmost importance that tasks are completed within the requested time frames, implying consideration of worst-case behavior in contrast to average-case behavior. Due to these considerations, the authors state that at the time of writing (2008), no high performance processor enables the exact calculation of a program’s execution time even at the machine code level.

For that reason, one is forced to attempt other approaches to obtain worst-case execution times in absolute units. The authors propose two different methods of measuring code execution time. The simplest method is to integrate signal generators into the source code. The signals trigger a high-resolution timer that can be started and stopped by writing into a register. The measuring device needs to be connected to the processor bus and placed into the system memory address space. Another method is possible for processors equipped with a JTAG interface. The standardized JTAG Boundary Scan Mechanism utilizes dedicated hardware connectors that make it possible to access the processor registers, control execution, set breakpoints and so on. An advantage with this approach is that it is non-intrusive and does not influence the timing behavior of the source code.
3.3 Summary

- To ensure the schedulability of all real-time tasks, transition times of state changes must be known. This information must be integrated into the schedulability test.
- WCET times must be calculated for the particular CPU and operating system that the application runs on. Analysis of frequency dependant execution times should also be performed due to the possibility of the existence of a non-linear relationship between frequency and execution time, e.g. due to changing memory frequency.
- If the target processor has an on chip high-resolution timer, signals can be integrated into the source code facilitating execution time measurement. If the target processor has a JTAG interface, the same measurements can be generated by using dedicated hardware on the processor without the need for integrating signals into the source code.
4 Power management techniques for real-time systems

This chapter aims to describe methods developed to address the challenges presented in the previous chapter. Various on-chip and off-chip power-saving technologies have been developed to address sources of power waste. Many are all hardware solutions such as smaller silicon process geometries, active well biasing and auto-idle detection circuits. Other technologies require software. This chapter will present an overview of a strategy for achieving energy savings that combine hardware circuitry with sophisticated software to achieve power savings; dynamic frequency voltage scaling. Dynamic frequency scaling refers to the possibility of reducing the clock frequency of the CPU during runtime, thus enabling the processor to operate at a lower voltage. Many different variants of the basic method have been put forward, as we will see. Most researchers formulate the problem of executing real-time tasks as an optimization problem with various constraints. The purpose of this chapter is to give the reader an overview of the solutions proposed and analyze the solutions collectively to discern what they require from an implementation perspective.

Chapter 5 will address what hardware support for dynamic frequency voltage scaling the i.MX31 ADS platform provides.

4.1 Sources of power consumption

Power consumption in complementary metal-oxide semiconductor (CMOS) integrated circuits (ICs) is broadly classified as dynamic power in a circuit while it is operating (e.g. switching), and static power while it is not operating but still powered (e.g non-switching steady state or transistor-off state). Static or leakage power dissipation also occurs when a circuit is operating, although for current CMOS processes, this is tiny compared to the dynamic power dissipation. However, as CMOS geometries continue to shrink, the static power will become ever more significant. Therefore, power-saving technologies must address both forms of power consumption to improve phone talk time, standby time and other power metrics of the appliance.

4.2 Dynamic Voltage Frequency Scaling

The power consumption of a micro-electric chip such as a CPU is dominated by the dynamic power dissipation $P_d$ of the CMOS transistors which is given by $P_d = C_{eff} \cdot V_{dd}^2 \cdot f$, where $C_{eff}$ is the effective switched capacitance and $f$ is the frequency of the clock. However, the gate delay $D$ of the
transistors is inversely related to the supply voltage as given by the formula \( D = k \cdot \frac{V_d}{(V_{dd}-V_t)^2} \), where \( k \) is a constant and \( V_t \) is the threshold voltage. Hence, in CMOS circuits, the cost of switching can be lowered by reducing the supply voltage at the price of a lower maximum attainable switching frequency.³

Many algorithms have been developed to exploit the situation described above. For example, the dynamic speed scaling problem without a sleep state was examined by Yao et al. (1995). They gave an optimal offline algorithm for the problem. They also defined a simple online algorithm called Average Rate (AVR)⁴. Bansal et. al. proposes another natural online algorithm called Optimal Available⁵. Irani et al. examine the problem of a DVFS system with a sleep state⁶.

According to Irani et al., there are a number of issues in the real-world problem of power management that is not incorporated in the above mentioned solutions. These have to do with the latency introduced by transitioning from one state to another. Another simplification of the solutions is that they assume a continuous power function. In reality, there are a finite number of speeds at which the system can run and the algorithm must ultimately select one of them. According to Irani, introducing realistic models makes the optimization problem much harder to solve and therefore much of the work on dynamic speed scaling makes these assumptions. As they put it, it remains to determine whether these assumptions are in fact reasonable.⁵

### 4.3 Summary

- Many proposed solutions to the DVFS problem make idealizations of the real situation to facilitate the arrival at an analytical solution. Implementation should facilitate testing of many different algorithms under realistic assumptions for selection of optimal method.
- It should be determined how the specific hardware of the i.MX31 ADS constrains the DVFS solution.
5 Hardware support for power management

The purpose of this chapter is to give an overview of the target hardware, the Freescale i.MX31 ADS, with special regards to built-in facilities that support DVFS. These facilities will be analyzed with regard to their suitability of use in a real-time environment.

The Freescale i.MX31 ADS is a multimedia development platform which consists of a baseboard, a CPU board and a power management board. The CPU board is equipped with an i.MX31 ARM11 MCU. The CPU board has a number of two-pin connections that can be used to measure the energy consumption in different parts of the system. The primary target of the i.MX31 processor is the mobile device market; thus, a lot of thought and effort has been invested to optimize performance and provide longer performance time for mobile systems based on the processor.

This chapter will focus on the following:
- i.MX31 support for frequency scaling.
- i.MX31 support for automatic frequency scaling and a discussion of the suitability of such a system in a real-time context.
- MC13783 support for voltage scaling.
- Allowed voltage levels for different frequency intervals.
5.1 i.MX31 support for frequency scaling

The Clock Controller Module (CCM) on the i.MX31 controls the system frequency, distributes clocks to various parts of the chip, controls the reset mechanism of the chip, and provides an advanced low-power management capability for the i.MX31 processor. The CCM includes these distinctive features:

- A PLL (Phase-locked loop) is a hardware feedback mechanism used to generate an output frequency that has a fixed relation to the phase of a reference signal. It will be utilized to generate frequencies in the range of 266 – 532 MHz from an external oscillator clocked at 26 MHz.
- Clock distributions – division of PLLs output clocks by post dividers PD:s.
- CCM registers that are writeable by system software to manipulate PLL and PD values.

The i.MX31 processor has three Digital PLLs in the system that generates three separate clock frequencies from the PLL reference clock. The PLL reference clock, in turn, can be generated either from an external high frequency source (CKIH) which is what is used on the i.MX31 ADS, or from a low frequency source that has been passed through a Frequency Pre-Multiplier (FPM).

There are three PLLs in the system that generates three separate clock frequencies from the PLL reference clock:

- The MCU PLL, configured by the MPCTL register, produces the mcu_main_clk clock. The MCU clock sub-domain is generated from that clock.
- The Serial PLL produces the reference clock to drive serial communication protocols.
- The USB PLL, configured by the UPCTL register, produces the clock for the USB circuitry (60 MHz).

The clock generation is described below in figure 8. Note that the mcu_clk is the MCU core clock and that its generation is highlighted in the figure. The default operation on the i.MX31 ADS is that the CKIH clock source is selected as the input source to the MCU PLL. The CKIH is an external oscillator clocked at 26 MHz. The output value of the MCU PLL is controlled by the MPCTL register. As can be seen in figure 8, the MCU PLL generates the mcu_main_clk or the main clock of the MMU. To generate the clock of the CPU core, the mcu_main_clk is passed through a post divider. This enables further frequency scaling from the original value. The post divider is controlled by the PDR0 register. The overall control of the frequency scaling is controlled by the PMRC0 register. Figure 7 presents an overview of the registers involved in frequency scaling.

Figure 7 Overview of the registers involved in frequency scaling
MCU PLL Control Register

The MCU PLL is set by writing to the MPCTL register.

Figure 8 System clock generation, MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor.

Figure 9 MCU PLL Control Register, MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor.
The output frequency is determined by the following formula where the binary representations of the parameter values are written to the control register above:

\[
F_{\text{ref}} \times 2^{\frac{MF_I + MF_N}{PD}} = F_{\text{vco}}
\]

Figure 10 MCU PLL Output frequency (Fvco), MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor.

As can be seen from the formula, the same frequency can be attained by different combinations of parameter values. The USB and Serial PLLs are set in a similar manner by writing to their respective registers.

Figure 11 PDR0 Post divider register 0, MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor.

In addition to changing the MCU PLL, a lower frequency for the mcu_clk can be achieved by using the mcu post divider. The PDR0 register is written to scale the post divider from 1 (write 000 to MCU_PODF) to 8 (write 111 to MCU_PODF).

Figure 12 Power management control register (PMCR0), MCIMX31RM Rev. 2.4, 12/2008 Freescale Semiconductor.
The PMCR0 register is the controlling register for frequency scaling on the i.MX31 platform. It enables software to write either both a new PLL and post divider value or just a new post divider value. It contains hardware that enables checking that the PLL has completed the transition to a new value.

The following bits control the frequency update procedure:

**DFSUP[1]**
- 1 for MCUPLL update

**DFSUP[0]**
- 0 pll and post-dividers update
- 1 post-dividers update only

**UPDTEN**
- 0 SW is not enabled to write new setting of frequency change because PLL is not locked yet.
- 1 SW is enabled to write new setting of frequency change.

---

**5.2 i.MX31 support for automatic DVFS**

In addition to manually scaling the voltage, the i.MX31 incorporates hardware to facilitate DVFS. Two separate approaches are supported, the reactive approach and the predictive approach. The two approaches are illustrated in the picture below.
The reactive approach is a hardware-driven mechanism with minimal software interference. In reactive mode the DFVS hardware measures system load. When it detects changes in performance requirements, it automatically adjusts the voltage and frequency of the CPU.

The predictive approach is a software-driven mechanism, where the i.MX31 DVFS hardware writes the system load measurements to the dedicated software-readable registers. The hardware also provides the interface for the software to program a pre-defined pattern for voltage and frequency. For extreme cases when there is a sudden surge in the system load, requiring maximum performance from the system, the panic mode of the DVFS activates, switching to the maximum frequency and voltage.
The system load register also outputs its value to the reactive logic. The reactive logic includes software-pre-programmed threshold system load levels. Once the system load changes by crossing a threshold boundary, the reactive logic sends a command to the DVFS interface to switch voltage and frequency. If the system load suddenly jumps to a pre-defined maximum performance threshold value, the system triggers the panic mode, switching as soon as possible to the highest voltage and frequency level. Software complexity involved in the reactive mode is negligible – primarily involving initial system setup and occasional adjustments during runtime. For the predictive mode, the software developer can implement as complex algorithm as desired, as the i.MX31 processor provide all the hardware tools for that.

Although the built in DVFS hardware mechanism of the i.MX31 processor has great potential for energy savings its use is unsuitable for real-time systems since it acts without any information of the task deadlines. For a real-time system, operating frequencies must be set by a scheduler aware of all the deadlines and timing characteristics of the system.

### 5.3 MC13783 support for voltage scaling

The MC13783 Power management IC contains a SPI interface port. The SPI port is configured to utilize 32-bit serial data words, using 1 read/write bit, 6 address bits, 1 null bit, and 24 data bits. The SPI's 64 registers correspond to the 6 address bits.

![Figure 16 MC13783 SPI Interface, MC13783GPLDRM Rev. 1.1, 4/2008, Freescale Semiconductor.](image1)

Bits 0-5 of register 24 is programmed to adjust the voltage level of SW1A, connected to the CPU power in pin. The available voltage levels are presented in the table below.

![Figure 17 MC13783 Register 24, MC13783GPLDRM Rev. 1.1, 4/2008, Freescale Semiconductor.](image2)
As can be seen from the table above, frequencies in the range of 0 – 400 MHz are attainable as long as the core voltage remains above 1.22 V. However to attain the higher frequencies, in the range 401 – 532 MHz the voltage must be set to more than 1.38 V. An interesting fact is that a range of voltages are allowed for each frequency span. However, one should keep in mind that no benefit accrues from running the processor at a higher voltage than the minimum. In fact, running the processor in “overdrive-mode” causes the IC to wear out and should be avoided. It should be clear from the above discussion that the voltage level requirements in relation to the CPU frequency of the i.MX31 are far from linear. It is important to note this fact, as many algorithms for DVFS consider the voltage/frequency relationship to be linear.
Figure 20 presents the allowed input and output frequency values for the MCU PLL. On the i.MX31 ADS the CKIH frequency is clocked at 26 MHz and is used as the input frequency of MCU PLL. This limits the only allowed pre-division factor (PD) value to 1 to keep the MCU PLL reference frequency in range. This in turn limits the attainable MCU PLL frequencies determined by the formula in figure 10. The lowest attainable MCU PLL frequency is thus 208 MHz. Any lower frequencies must be produced by scaling this value using the post divider register.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase lock time</td>
<td>---</td>
<td>---</td>
<td>100</td>
<td>µs</td>
<td>In addition to the frequency</td>
</tr>
<tr>
<td>Frequency lock time (FOL mode or non-integral MF)</td>
<td>---</td>
<td>---</td>
<td>398</td>
<td>---</td>
<td>Cycles of divided reference clock.</td>
</tr>
</tbody>
</table>

Figure 21 PLL switch times, MCIMX31_5 Rev. 4.2, 11/2008, Freescale Semiconductor.

Figure 21 presents the switching times when switching to a new MCU PLL reference frequency. The time of $100 \, \mu s + 398$ cycles of divided reference clock is unacceptably high for our target real time system. What is even worse, measurements done of switching times done in the preliminary stages of this thesis work reveal that the tick counter is not updated during the MCU PLL switch, thus leaving the MMU core in a temporally undetermined state when the switch is completed. This is unacceptable for any real time system and the method of switching PLL values during runtime to scale frequencies must therefore be rejected.

### 5.5 Summary

- The Freescale i.MX31 has various built in mechanisms that facilitate power management. To relieve the operating system of unnecessary work, such hardware support should be utilized as far as possible.
- Clock scaling is made possible by writing to the MCU PLL control register. The transition is not instantaneous new PLL values should not be written while the CPU is executing programs with real time constraints due to loss of temporal integrity.
- A more suitable method of updating the frequency of the CPU is by setting the PLL to the highest value needed to be attained by the system. Lower frequencies can then be accessed by using the post divider register.
- The voltage can be scaled by writing to the PMIC which is physically connected to the i.MX31. The time to perform this scaling must be determined.
- The voltage/frequency relationship of the i.MX31 is far from linear as only two voltage levels, 1.22 and 1.38 V, are needed to gain access to all attainable frequencies. Effort should be directed towards investigating how the energy consumption changes with the frequency if the voltage stays the same. If the energy consumption per clock cycle is the same it would further reduce the complexity of the DVFS algorithm as it would only consist of transitioning between two voltage/frequency pairs.
6 Supporting software platforms

The software environment in which a real-time application runs can have a significant impact on the performance of the system. The goal of this chapter is to examine how the choice of software system effects the implementation of a real-time power manager. Variables such as scheduling algorithm and hardware access will be of high interest.

6.1 U-boot 1.3.3

A boot loader, sometimes referred to as a boot monitor, is a small piece of software that executes soon after powering up a computer. On a desktop Linux PC this program is often LILO or GRUB, which resides on the master boot record (MBR) of the hard drive. After the PC BIOS performs various system initializations it executes the boot loader located in the MBR. The boot loader then passes system information to the kernel and then executes the kernel. For instance, the boot loader tells the kernel which hard drive partition to mount as root.9

In an embedded system the role of the boot loader is more complicated since these systems do not have a BIOS to perform the initial system configuration. The low level initialization of microprocessors, memory controllers, and other board specific hardware varies from board to board and CPU to CPU. These initializations must be performed before an operating system kernel image can execute9.

At a minimum an embedded loader provides the following features:

• Initializing the hardware, especially the memory controller.
• Providing boot parameters for the operating system kernel.
• Starting the operating system kernel

Additionally, most boot loaders also provide “convenience” features that simplify development:

• Reading and writing arbitrary memory locations.
• Uploading new binary images to the board's RAM via a serial line or Ethernet
• Copying binary images from RAM to FLASH memory

The i.MX31 ADS board comes with a pre-installed version of the boot loader U-boot version 1.3.3. When the power is switched on, U-boot initializes the i.MX31 ADS board and provides serial and Ethernet communication to a host development platform, usually a desktop computer. Commonly, U-boot is then used to transfer the compiled operating system kernel image over the Ethernet link using the FTP protocol. However, it can also be used to load simple binary executables. This presents an interesting option for close control of the hardware for development proposes. In addition this setup can also be used for measurement purposes since the developer has complete control of all code that executes on the target platform.9

6.2 OSE 5.4

OSE 5.4 is a real-time operating system (RTOS) developed by Enea designed for fault-tolerant, distributed systems. One of the primary markets for OSE is mobile platforms, specifically handling the communication protocol with the base station. The fundamental building block in OSE is the
process. The processes execute in parallel and can preempt each other. Five different process types are available in OSE 5.4:

**Interrupt processes** are invoked on hardware interrupts or when summoned by another process. They can only be preempted by a higher priority interrupt process.

**Timer-interrupt processes** are similar to interrupt processes but are invoked periodically by the system timer. A timer interrupt process runs with the same priority as the system timer interrupt process.

**Prioritized processes** are the most common process type. A prioritized process has to be assigned a priority when created. They are usually created as infinite loops to receive signals and perform a certain task.

**Background processes** run when no interrupt, timer-interrupt or prioritized process is ready. Background processes are scheduled using time slicing. Each background process is assigned a time when created.

**Phantom processes** are not real processes. They are never scheduled and contain no code. Phantom processes are used to redirect signals in distributed systems.

### 6.3 OSE Bios interface

OSE Bios offers basic services for use of the System Call mechanism. bios.h is part of the System Programming Interface (SPI), which is mainly intended for platform developers. Bios handles the system call mechanism. The system call mechanism allows programs to call the executive or other components that have registered with Bios, without knowing the address that the executive or component is located on. It also provides a way for user mode code to switch to supervisor mode in a controlled way.

The OSEBios component has nothing to do with the Basic Input/Output System (BIOS) term in the desktop PC world.

Whenever an application executes a system call instruction, the execution jumps to the Bios component. Bios provides three operations that are essential for the operation of the system; Install, Open and Call. The bios services are available for all types of processes. Installation operations require Supervisor Mode privileges.

```c
long biosInstall(const char *name, BiosFunction *entrypoint, unsigned long flags);
```

Installs a function with the specified name string. The installed function will be available for programs via the biosOpen and biosCall functions.

```c
unsigned long biosOpen(const char* name);
```

Opens a bios function for use. The function with the given name is located and a handle to it is returned. The handle is used to call the function via biosCall.

```c
long biosCall(unsigned long handle, ...);
```

Calls the function specified by the handle with the specified parameters (up to seven of them).
6.4 Summary

- The U-boot boot loader is a suitable platform for development of hardware driving software and for measurement of power consumption.
- OSE supports static priority scheduling. The real-time scheduling algorithm appropriate for such a system is rate-monotonic scheduling.
- The OSE Bios framework should be used to implement the DVFS functionality to ensure access to hardware from user space processes.
7 Design description

Consistent with the delimitation of this thesis, the high-level DVFS scheduler is not designed as a part of this study. It should however be noted that due to the clean separation between hardware and software, the overall system can still be tested and its performance assessed. A further benefit of this is that many different algorithms for PM can easily be tried out.

The DVFS module is implemented using the OSE Bios framework. The module registers its interface functions with Bios at startup. From this point, all processes can gain access to the DVFS functionality by issuing the appropriate bios calls.

7.1 Voltage scaling

Supported voltage levels: VOLTAGE_120, VOLTAGE_140, VOLTAGE_160.

The voltage scaling module is used to scale the supply voltage of the CPU by allowing OSE processes to set the supply voltage of the i.MX31 by providing a signaling interface. The signals are received by the dvfs_slave process which initiates the appropriate bioscall. The bioscall initiates a communication session with the MC13783 PMIC over SPI. The SPI message causes the register controlling SW1A, connected to the power in pin of the CPU, to be updated causing the output of a new voltage value. For a CPU frequency of 532 MHz, the minimum voltage level is 1.40 V. A voltage level of 1.60 V is considered overdrive and should be avoided.
7.2 Frequency scaling

Supported frequency values: FREQ_532, FREQ_399, FREQ_200, FREQ_100.
Supported modes: PLL_UPDATE, NO_PLL_UPDATE.

The frequency scaling module allows OSE processes to set the CPU frequency of the i.MX31 via a signal interface. The signals are received by the dvfs_slave process which initiates the appropriate bioscall. The DVFS Bios module processes the bioscall and sets the frequency of the CPU by writing to the PLL, post divider and frequency control registers. If the calling OSE process desires it, the dvfs_slave process returns the frequency at the time of the requested frequency change. This enables processes to “clean up” after themselves when having completed their execution. They accomplish this by requesting a new frequency change at the end of their task termination to restore
the old frequency value. This greatly simplifies management of frequency scaling in situations where processes are set to run at different frequencies with preemption enabled. The PLL must be updated when switching to or from 532 MHz. The other frequencies use a PLL value of 399 MHz.

### 7.2.1 Debug functions

In addition to the main functionality implemented by the OSE DVFS module, two debug utilities have been developed:

```c
blink_debugleds();
```

Blinks debugleds on the i.MX31 ADS mainboard. Used to visually confirm frequency changes as the leds blink at a speed proportional to the MCU frequency.

```c
blink_pmic_leds(int blink_speed);
```

Blinks a led on the MC13873 PMIC board. Used to confirm contact with the PMIC. Supported values: NO_BLINK, SLOW_BLINK, FAST_BLINK.
8 Test suite

The test suite consists of two different parts. Tests 1-6 are tests performed to fully determine the power/current consumption and timing characteristics of the processor to enable implementation of a real-time DVFS scheduling algorithm described in the ENEA internal design description document Doc ID 10727, Revision 4. Test 7 probes the effectiveness of the DVFS system developed for this master thesis used in conjunction with that real-time DVFS scheduling algorithm.

8.1 Test cases

8.1.1 Test case 1: CPU 100% load

**Description:** Run the CPU at 100% load in OSE. Measure the voltage drop across 0.5 Ω resistor R22 on the i.MX31 ADS development board as the MCU PLL is programmed with new frequency values in conjunction with setting the supply voltage to the values in the following table:

<table>
<thead>
<tr>
<th>Supply voltage (V)</th>
<th>MCU PLL frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.60</td>
<td>532</td>
</tr>
<tr>
<td>1.40</td>
<td>532</td>
</tr>
<tr>
<td>1.40</td>
<td>399</td>
</tr>
<tr>
<td>1.20</td>
<td>399</td>
</tr>
<tr>
<td>1.20</td>
<td>266</td>
</tr>
<tr>
<td>1.20</td>
<td>200</td>
</tr>
<tr>
<td>1.20</td>
<td>133</td>
</tr>
<tr>
<td>1.20</td>
<td>100</td>
</tr>
<tr>
<td>1.20</td>
<td>66</td>
</tr>
<tr>
<td>1.20</td>
<td>33</td>
</tr>
</tbody>
</table>

**Expected output:** The voltage drop is expected to vary linearly while keeping the supply voltages constant.

8.1.2 Test case 2: CPU 0% load

**Description:** Have U-boot idle. Measure the voltage drop across 0.5 Ω resistor R22 on the i.MX31 ADS development board as the MCU PLL is programmed with values in conjunction with setting the supply voltage to the values in the following table:

<table>
<thead>
<tr>
<th>Supply voltage (V)</th>
<th>MCU PLL frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.60</td>
<td>532</td>
</tr>
<tr>
<td>1.40</td>
<td>532</td>
</tr>
<tr>
<td>1.40</td>
<td>399</td>
</tr>
<tr>
<td>1.20</td>
<td>399</td>
</tr>
<tr>
<td>1.20</td>
<td>266</td>
</tr>
<tr>
<td>1.20</td>
<td>200</td>
</tr>
<tr>
<td>1.20</td>
<td>133</td>
</tr>
<tr>
<td>1.20</td>
<td>100</td>
</tr>
<tr>
<td>1.20</td>
<td>66</td>
</tr>
<tr>
<td>1.20</td>
<td>33</td>
</tr>
</tbody>
</table>
Expected output: The voltage drop is expected to vary linearly while keeping the supply voltages constant. It is also expected that the voltages are shifted to lower values for the corresponding frequency/voltage settings compared to test 1 & 2.

8.1.3 Test case 3: PLL vs. Post divider scaling

Description: Have OSE idle. Set the MCU PLL to 399 MHz and the supply voltage to 1.20 V. Scale the MMU frequency to 200 MHz and 100 MHz using the post dividers 2 and 4. Measure the voltage drop across 0.5 Ω resistor R22 on the i.MX31 ADS development board.

Expected output: The voltage drop values attained by scaling the frequency using the post divider is expected to be larger then the values obtained from test 2.

8.1.4 Test case 4: Power consumption while switching frequency

Description: Set MCU PLL to 399 MHz and switch repeatedly between 399 and 200 and 100 MHz by changing the post divider value to 1, 2 and 4. Measure the voltage drop across 0.5 Ω resistor R22 on the i.MX31 ADS development board. Graphically display the resulting value as a function of time. The purpose of this test is to detect any increase in current/power consumption during the switch from one frequency to another.

8.1.5 Test case 5: Timing characteristics of frequency scaling

Description: Set MCU PLL to 399 MHz and switch repeatedly between 399 and 200 and 100 MHz by changing the post divider value to 1, 2 and 4. Use the OSE system function get_systime() to get the tick and microsecond values before and after the switch. Plot the difference in values graphically. The purpose of this test is to determine the average and worst case delays in switching to a different frequency using the post dividers to enable the technique to be used in a real-time context.

8.1.6 Test case 6: Timing characteristics of voltage scaling

Description: Program the MC13783 PMIC to switch SW1A (connected to the CPU power in pin) between the voltages 1.60, 1.40 and 1.20 V. Plot the result as a function of time. The purpose of this test is to determine the average and worst case delays in switching to a different supply voltage to enable the technique to be used in a real-time context.

8.1.7 Test case 7: Power consumption of integrated system

Description: Integrate the DVFS system with the real-time DVFS scheduling algorithm described in the internal ENEA internal design description document Doc ID 10727, Revision 4. Set the process times to 1 ms and the periods to 4, 6 and 8 ms. Test the system with the following settings:
<table>
<thead>
<tr>
<th>Mode</th>
<th>Supply voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>All processes run at 532 MHz.</td>
<td>1.60</td>
</tr>
<tr>
<td>All processes run at 532 MHz.</td>
<td>1.40</td>
</tr>
<tr>
<td>All processes run at 399 MHz.</td>
<td>1.40</td>
</tr>
<tr>
<td>All processes run at 399 MHz.</td>
<td>1.20</td>
</tr>
<tr>
<td>Idle runs at 100 MHz</td>
<td>1.20</td>
</tr>
<tr>
<td>Process 3 is dynamically scaled during the hyper period.</td>
<td>1.20</td>
</tr>
</tbody>
</table>

The purpose of this test is to determine the robustness of the OSE DVFS module and the energy effectiveness of the integrated real-time DVFS system.
9 Results

In this chapter the results of the tests described in the previous chapter are described. The tests aim to answer the questions raised in the initial chapter. Please see appendix A for the raw data from these tests.

9.1 Determination of CPU power consumption and timing characteristics

This section presents results that precisely determine the power and timing characteristics of all proposed voltage and frequency scaling methods.

9.1.1 Power and current consumption as a function of the voltage and frequency level

This section aims to precisely determine the power/current characteristics of all relevant frequency voltage settings for the i.MX31.

9.1.1.1 Test case 1: CPU 100% load and Test case 2: CPU 0% load

To enable effective frequency and voltage scaling, the power and current consumption characteristics must be precisely determined as a function of the frequency/voltage combination. To achieve this, test one and two are performed. For test one, the processor is made to run at a load of 100 % and the frequency is scaled by writing values to the MCU PLL in the range of 532 to 33 MHz. The supply voltage of the CPU is altered from 1,60 to 1,40 V. For test two, the same voltage/frequency combinations are tested but this time the CPU is made to run at a load of 0 % by idling in u-boot. The voltage drop over the R22 resistor on the i.MX31 ADS is measured and the current and power consumption is computed.

![Figure 22 Current Consumption OSE/U-boot i.MX31 ADS](image-url)
The voltage drop across the resistor is divided by the resistor’s resistance \((0.5 \, \Omega)\) to calculate the current values through the resistor according to \(I = \frac{U}{R}\). As can be seen from the graph, the current consumption relationship to the frequency setting is linear as long as the voltage is kept constant.

![Power Consumption OSE/U-boot i.MX31 ADS](image)

Figure 23 Power Consumption OSE/U-boot i.MX31 ADS

The current values are multiplied with the supply voltage to the CPU to generate the dissipated power from the CPU according to \(P = U \cdot I\). As for the current, the power consumption relationship to the frequency setting is linear as long as the voltage is kept constant.

### 9.1.1.2 Test 3: PLL vs. Post divider scaling

The frequency of the MMU Core can be scaled both by adjusting the MCU PLL value and by changing the post divider as described in chapter four. To enable the post divider method for use in power management, the current consumption for this technique must be determined.

![Current consumption OSE Idle PLL vs PD scaling](image)

The graph plots the current consumption measured by scaling frequencies by adjusting the MCU PLL versus setting the MCU PLL to a constant of 399 MHz and adjusting the post divider. As can be seen in the plot, current consumption is slightly higher when using the post divider method but not catastrophically so. Linear regression is used to calculate the difference between having OSE idle at 100 MHz using a PLL value of 100 MHz versus a value of 399 MHz and a post divider of 4. The
difference is calculated to 0.62 mA, a very small price to pay for retaining the temporal integrity of the MMU core.

9.1.1.3 Test 4: Power consumption while switching frequency

To enable the use of frequency scaling with post dividers for use in power management, the power consumption during the switch from one frequency setting to the next must be determined. This information can then be utilized in determining the optimal power management strategy in a given situation. The voltage drop over the R22 resistor on the i.MX31 ADS is measured and the resulting values are plotted as a function of time. The goal of this test is to detect any anomalous readings at the moment of the frequency switch. The CPU is given the instruction to switch frequency level at given intervals and then given a 100 % CPU load in between switches to provide a reference current consumption reading.

As can be seen from the graphs, no increase in voltage drop can be detected in the switching moment. It appears as if the work performed by the CPU consumes the same amount of current as the processing of normal CPU tasks at a certain voltage/frequency setting.
9.1.2 Timing characteristics of frequency/voltage scaling

To enable frequency and voltage scaling of the i.MX31 to be used in a real-time environment, specific timing characteristics must be determined for all state transitions involved.

9.1.2.1 Test 5: Timing characteristics of frequency scaling

The average and worst case switch times must be determined frequency scaling. In this test, the post divider is used to scale the frequency between 100, 200 and 399 MHz. A total of 1000 iterations are performed for each frequency switch and the resulting switch times are displayed graphically.
The graphs above summarize the results of the tests. The switching times are in the range of 4 – 50 µs. This should be compared to the tabulated values of PLL lock times presented in chapter 4 of roughly 500 µs. It should also be emphasized that the system tick values are continually updated during the frequency switch when performed with the post dividers, retaining the system temporal integrity.

An interesting fact evident by the graphs above is that the worst case switch times are sometimes a factor of 10 longer then the average times. A speculation is that either something internal in the ARM core of the i.MX31 or a system process in OSE is responsible for this behavior. As can be seen from the graphs, the worst case times are registered only sporadically. This means that to ensure that all deadlines are met; very conservative estimates of switching times should be used. If however, deadlines can be allowed to be overrun sporadically, a much heavier CPU load can be scheduled using the average switch times.
9.1.2.2 Test 6: Timing characteristics of voltage scaling

The switch times for voltage scaling must be determined. The supply voltage of the i.MX31 is measured as a new voltage setting is requested from the MC13783 PMIC and the result is displayed graphically as a function of time.

![Figure 27 Switch from 1.20 V to 1.40 V](image)

![Figure 28 Switch from 1.40 V to 1.20 V](image)

![Figure 29 Switch from 1.40 V to 1.60 V](image)

![Figure 30 Switch from 1.60 V to 1.40 V](image)

From the plots above the switch times are determined to be approximately 2 ms for each voltage switch. This value is rather high and voltage cannot be scaled using this technique in a running real-time system with process execution times on the order of milliseconds. For our target real-time system such a value is unacceptably high. Therefore, in our target system, the voltage level must therefore be set prior to the start of the program with real-time constraints.
9.2 Verification of OSE DVFS module

The goal of this master thesis is to precisely determine the power/timing characteristics of the i.MX31 platform and to develop the capability to dynamically scale the voltage and the frequency of the chip during runtime. The ultimate aim of this work is to enable the implementation of a real-time DVFS scheduler described in the ENEA internal design description document nr. 555555. Test 7 verifies the functionality of the OSE DVFS module in conjunction the real-time DVFS scheduler. In addition, the test determines the power/current savings achieved by the integrated system.

9.2.1.1 Test 7: Power consumption of integrated system

The integrated system runs three 1 ms long processes with the periods 4, 6 and 8 ms scheduled according to the rate monotonic principle. The progression of the test is described step by step below:

- The i.MX31 starts out at the default settings of a CPU frequency of 532 MHz and a supply voltage of 1.60 V.
- The system detects the frequency level and adjusts the core voltage to the optimal level for this frequency, which is 1.40 V.
- A schedulability test is then performed by the real-time scheduler who determines that at this CPU load, all processes will meet their deadlines at a CPU frequency of 399 MHz. The MCU PLL value is updated with the new value.
- The system detects the change in operating frequency and adjusts the voltage to the optimal level for this PLL value which is 1.20 V. To further reduce
- The frequency is dynamically scaled to 100 MHz using the post dividers when the system is idle.
- The execution frequency of process 3, the least prioritized process, is dynamically lowered until deadline overruns are detected. This probing enables the lowering of frequencies beyond the frequency calculated by the static schedulability test which assumes worst case behavior in estimation of system parameters.

![Current consumption DVFS i.MX31](image)

**Figure 31 Current consumption integrated DVFS system**
The measured results from each operating point are showed in the two figures above. The decrease in both power and current consumption is quite dramatic when it is kept in mind that the integrated real-time DVFS system delivers exactly the same performance as the system running at the maximum rating. That is, all tasks are executed according to their specified periods meet their respective deadlines. To put the above results in perspective, a simplified battery lifetime calculation using a standard Li-Polymer 950 mAh cell phone battery (e.g. Sony Ericsson BST-33) can be performed. If current consumption can be approximated to be constant during the batteries charge cycle the following increase in battery life can be calculated. Thus, a cell phone could run the test system almost three days compared to only one using the default settings while delivering exactly the same quality of service.
10 Discussion

The DVFS system in combination with the real-time DVFS scheduler shows remarkable current/power savings when allowed to run at optimal levels. It should be noted however that the voltage level as well as the MCU PLL setting must be determined prior to starting the actual execution of the real-time tasks. This level is best determined by calculating the schedulability of the system using the worst case switching times added to the execution times of the real-time tasks. Once this level is found to be acceptable from a deadline overrun perspective, further frequency scaling can be employed using the post divider register. This method ensures quick, deterministic frequency switching with only a slight current/power penalty versus scaling the frequency with the PLL.

10.1 Summary of results

This section will present answers to the questions presented in section 1.2.

10.1.1 Determination of CPU power consumption vs. performance

How does the power consumed by the processor depend on the frequency settings?

- The power consumed increases linearly with higher frequencies as long as the supply voltage is kept constant. The specific values can be found in the results section. When the voltage is increased, the power consumption slope is increased but the overall relation remains linear. If the frequency is scaled by using post dividers, slightly higher current/power consumption is the result versus setting the frequency to the corresponding values using the PLL.

What are the timing characteristics of the proposed frequency and voltage scaling methods?

- The timing characteristics of the frequency and voltage scaling methods have been precisely determined and presented in chapter 8. Voltage scaling using the post divider method is considered superior to the PLL scaling method due to faster switching times and retained system temporal integrity.

What hardware functionality does the i.MX31 ADS have to support power management and what is its suitability of use in a real-time system?

- The MCU clock can be scaled by adjusting the output of the MCU PLL. This method should only be used prior to starting the real-time execution of the application program. This is due to the relatively long switch times and the temporal indeterminism that results from loosing the tick counter.

- The built in hardware load monitoring and frequency adjustment capabilities of the i.MX31 was found unsuitable for hard real time systems due to the possibility of missed deadlines. Any real-time PM should explicitly set frequency/voltage levels so as to be able to integrate the transitions into the schedulability tests.

- During execution of real-time tasks, the frequency should be scaled by using the post divider register. This allows for quick, deterministic switching of the frequency setting.

- The voltage can be adjusted by programming the MC13783 PMIC circuit. Due to long switching times the appropriate voltage level should be set prior to staring any real-time tasks. The voltage level should be the lowest allowed for the selected PLL frequency.
10.1.2 Implementation of Dynamic Voltage Frequency Scaling (DVFS) scheduler

What is required of the system to enable the implementation of power management in a real-time system?

- The power consumed must be known for all frequency/voltage states. This has been determined and tabulated in the results section.
- Transition times and the power consumption involved when transitioning between states. This has been determined and tabulated in the results section.

How can the DVFS scheduler be implemented in OSE?

- OSE implements fixed priority scheduling. It is therefore appropriate to assume that tasks will be scheduled according to the Rate Monotonic principle.
- The real-time DVFS scheduler will set the appropriate voltage and frequency values using a DVFS module as a part of this master thesis using a signal interface to a dvfs_slave process. The dvfs_slave process executes the appropriate bioscalls to execute the voltage/frequency scaling request. The DVFS module registers its support functions with the OSE bios interface at startup using a system hook.

10.2 Requirements

A compilation of the requirements, derived from the second chapter, including their status is presented below:

SWREQ1: The system should be designed in a manner that minimizes the energy it consumes.
- The power/current reduction of the integrated real-time system has been demonstrated in the results section.
- PASSED: TEST CASE 7

SWREQ2: The CPU should perform its required computations in the most energy efficient manner possible while observing all other requirements.
- The integrated DVFS system ensures that all real-time tasks are met while power/current consumption is minimized.
- PASSED: TEST CASE 7

SWREQ3: The CPU should utilize all appropriate on-chip hardware resources for power management to reduce power management overhead.
- The frequency switching is performed in the fastest, most robust manner possible by using the post divider register to switch between frequencies while utilizing the same PLL value.
- PASSED: TEST CASE 7

SWREQ4: The system must meet the deadlines of all real-time tasks.
- The DVFS real-time scheduler uses the worst case transition times of the frequency switches to ensure that all deadlines are met of hard real-time tasks.
- PASSED: TEST CASE 7
10.3 Future Work

In light of the previous chapter, the following areas should be pursued:

- To further reduce power consumption of running real-time tasks, methods such as clock gating and transitioning unused parts of the chip to low power states should be researched. In addition, investigation of the idle process in OSE should be performed to determine methods of reducing the power consumed while the CPU is idle.
- To reduce power consumption in general, when real-time issues are not important, work should be performed to enable the transition of CPU to low power states.
11 Appendix A

This appendix contains the raw data generated by the tests described in chapter 8.

11.1 Test 1: CPU 100% load & Test 2: CPU 0% load

<table>
<thead>
<tr>
<th>Supply voltage (V)</th>
<th>Freq (MHz)</th>
<th>CPU load 0% (mA)</th>
<th>CPU load 100% (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,60</td>
<td>532</td>
<td></td>
<td>36,18</td>
</tr>
<tr>
<td>1,60</td>
<td>399</td>
<td></td>
<td>29,42</td>
</tr>
<tr>
<td>1,40</td>
<td>532</td>
<td>18,78</td>
<td>32,36</td>
</tr>
<tr>
<td>1,40</td>
<td>399</td>
<td>15,12</td>
<td>26</td>
</tr>
<tr>
<td>1,20</td>
<td>399</td>
<td>11,72</td>
<td>20,54</td>
</tr>
<tr>
<td>1,20</td>
<td>266</td>
<td>8,8</td>
<td>15,36</td>
</tr>
<tr>
<td>1,20</td>
<td>133</td>
<td>5,98</td>
<td>10,22</td>
</tr>
<tr>
<td>1,20</td>
<td>66</td>
<td>3,84</td>
<td>7,58</td>
</tr>
<tr>
<td>1,20</td>
<td>33</td>
<td>3,04</td>
<td>6,14</td>
</tr>
</tbody>
</table>

11.2 Test 3: PLL vs. Post divider scaling

<table>
<thead>
<tr>
<th>Supply voltage (V)</th>
<th>MCU PLL frequency (MHz)</th>
<th>PLL scaling (mV)</th>
<th>PLL 399 MHz (mV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,20</td>
<td>399</td>
<td>20,06</td>
<td></td>
</tr>
<tr>
<td>1,20</td>
<td>266</td>
<td>14,92</td>
<td>20,06</td>
</tr>
<tr>
<td>1,20</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,20</td>
<td>133</td>
<td>9,90</td>
<td>12,64</td>
</tr>
<tr>
<td>1,20</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,20</td>
<td>66</td>
<td>7,14</td>
<td>9,20</td>
</tr>
<tr>
<td>1,20</td>
<td>33</td>
<td>5,88</td>
<td></td>
</tr>
</tbody>
</table>

11.3 Test 5: Timing characteristics of frequency scaling

<table>
<thead>
<tr>
<th>Mode</th>
<th>Average time (µs)</th>
<th>Worst case time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>399-&gt;200</td>
<td>4,31</td>
<td>52</td>
</tr>
<tr>
<td>399-&gt;100</td>
<td>5,31</td>
<td>31</td>
</tr>
<tr>
<td>200-&gt;399</td>
<td>5,87</td>
<td>34</td>
</tr>
<tr>
<td>200-&gt;100</td>
<td>7,27</td>
<td>53</td>
</tr>
<tr>
<td>100-&gt;399</td>
<td>9,42</td>
<td>56</td>
</tr>
<tr>
<td>100-&gt;200</td>
<td>9,85</td>
<td>56</td>
</tr>
</tbody>
</table>
### 11.4 Test 7: Power consumption of integrated system

<table>
<thead>
<tr>
<th>Mode</th>
<th>Supply voltage</th>
<th>Current consumption (mA)</th>
<th>Power consumption (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All 532, 1.60 V</td>
<td>1.6</td>
<td>36.6</td>
<td>58.6</td>
</tr>
<tr>
<td>All 532, 1.40 V</td>
<td>1.4</td>
<td>32.9</td>
<td>46.1</td>
</tr>
<tr>
<td>Idle 399, 1.40 V</td>
<td>1.4</td>
<td>26.9</td>
<td>37.7</td>
</tr>
<tr>
<td>Idle 399, 1.20 V</td>
<td>1.2</td>
<td>20.7</td>
<td>24.9</td>
</tr>
<tr>
<td>Idle 100, 1.20 V</td>
<td>1.2</td>
<td>15.5</td>
<td>18.6</td>
</tr>
<tr>
<td>Dynamic P3, 1.20 V</td>
<td>1.2</td>
<td>14.4</td>
<td>17.2</td>
</tr>
</tbody>
</table>
12 Bibliography


5 Bansal et. al. *Speed scaling to manage energy and temperature*, J. ACM 54, 1,1, 2007.


