Power models of system-on-chip and external memory for multimedia applications

Chinta, A.

Award date:
2006

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

POWER MODELS OF SYSTEM-ON-CHIP AND EXTERNAL MEMORY FOR MULTIMEDIA APPLICATIONS

By
Anuradha Chinta

Supervisors:
Dr. Ir. Marc Geilen
Ms. DRS. EFM Steffens

Eindhoven, November 2006

ACKNOWLEDGEMENTS

I would like to gratefully acknowledge the enthusiastic supervision of Dr. Ir. Marc Geilen, Assistant Professor at the Electrical Engineering department of Technical University Eindhoven and Ms. DRS.EFM Steffens**,** Senior Research Scientist at Philips Research Labs, Eindhoven for their invaluable technical guidance and support provided during this thesis work. I would like to thank them for providing me with an opportunity to carry out my M.Sc project at Philips Research Labs, Eindhoven.

I would like to thank Ms. LIC.CM Otero Perez at Philips Research Labs, Eindhoven for helping me with the tools and clarifying my doubts patiently. I would like to thank Mr. Ir. AT Burchard at Philips Research Labs, Eindhoven for his invaluable time and help with measurements, analysis and numerous stimulating discussions.

I am also thankful to Alina**,** PhD student for clearing my doubts. I wish to thank my classmates in TU/e for their help and support during my study program.

I thank my husband Subrahmanya Kumar and our families for their constant encouragement and moral support throughout my study period.

A. ABSTRACT

The multimedia functionality in modern handheld devices is computationally intensive and requires a lot of energy. The source of energy in these devices is usually a battery. Battery technology has not improved at the pace of increase in energy requirements. An important requirement in these devices is to reduce power consumption while meeting the timing constraints of multimedia applications. Multimedia applications have the advantage of allowing run time trade-offs between (picture) quality and power consumption. The (picture) quality and hence the power consumption can be controlled by application parameters. In order to allow for run time trade-offs, one needs to have an estimate of power consumption for various application parameter settings.

This thesis focuses on developing power models for a System-on-Chip and Double Data Rate (DDR) memory from application parameters. The power models were developed through physical measurements on a Philips PNX1500 platform by running MPEG-4 decoder application. We present two methods to develop power models. The first method develops power models at a higher abstraction level by excluding architecture level details, whereas the second method considers architecture level details. Power models developed with the first method are abstract and easy to model, but the validity of the models is limited to this specific platform. The architecture level details of the second method can be projected to other platforms. In this thesis, we also show that the architecture level details can be used to derive the scalable frequencies for dynamic frequency scaling method. The experiments with two input streams of different content suggest that the content of the input stream has no influence on power models.

B. TABLE OF CONTENTS

C. ABBREVIATIONS

4CIF	4 times Common Intermediate Format
A/D	Analogue to Digital
API	Application Programmers Interface
ASCII	American Standard Code for Information Interchange
BETSY	Being on Time Saves energY
BIS	Boot Info Structure
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CIF	Common Intermediate Format
DCS	Device Control and Status network
DDR SDRAM	Double Data Rate Synchronous Dynamic Random Access Memory
DMA	Direct Memory Access
DVP	Digital Video Platform
EJTAG	Enhanced JTAG
HD	High Definition
IEEE	Institute of Electrical and Electronics Engineers
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
MBS	Memory Based Scalar
MMI	Main Memory Interface
MMIO	Memory Mapped Input Output
MPEG	Moving Picture Experts Group
MPTK	Media Processing Tool Kit
NDK	Nxperia Developers Kit
OSAL	Operating System Abstraction Layer
PC	Personal Computer
PCI	Peripheral Component Interconnect
PMAN	Pipelined Memory Access Network
PNX	Philips NXperia
PSOS	Portable Scalable Operating System
QVCP	Quality Video Composition Processor
QCIF	Quarter Common Intermediate Format
RMSE	Root Mean Square Error
RS232	Recommended Standard232
SoC	System-on-Chip
TM	TriMedia
URD	Universal Register Debugger
VGA	Video Graphics Array

D. LIST OF FIGURES

E. LIST OF TABLES

F. LIST OF GRAPHS

1 Introduction

1.1 Problem description

Modern handheld devices incorporate a lot of multimedia functionality. Multimedia functions like video encoding and decoding are computationally intensive and cause a lot of energy consumption [1]. The source of energy in the handheld devices is usually a battery. Battery technology has not improved at the pace of increase in energy requirements [2]. Moreover, in handheld devices the capacity of the battery is limited. The battery is small and cannot be enlarged because of the restricted size and weight of the handheld. Therefore, in handheld devices reducing energy consumption while meeting the timing constraints of multimedia applications is an important requirement.

The energy consumption of a system can be reduced through various parameters of the application and architecture, and through different power management techniques like clock gating and dynamic voltage and frequency scaling. In multimedia applications, it is possible to make trade-offs between picture quality and energy consumption. For example, with a given battery, some times it is necessary to make sure that the battery operates long enough for a particular activity like playing video. In this case, we can extend the battery life by reducing the energy consumption with a compromise in picture quality but still meeting the timing constraints. The picture quality can be controlled by various parameters of the application. In order to allow run time trade-offs, we need to have an estimate of the energy consumption for various application parameter settings.

The goal of this thesis is twofold: (1) To develop models that can predict the power consumption of a system from application parameters without considering the architecture level details. In order to make fast, run time decisions, these models should be simple and abstract enough. The modeling effort required for these models is limited because they exclude all the architecture level details. But the validity of these abstract models is restricted to a specific context, for example to a specific platform. (2) To develop models that can predict the power consumption of a system from application parameters by considering the architecture level details. These models include all the platform parameters that influence the power consumption. These platform parameters can be used to estimate platform parameters in similar multimedia platforms. The parameters can also be used to investigate the compositionality in applications. Accurate estimates of the power consumption require capturing all the platform parameters that influence the power consumption. On the other hand, sometimes it is not possible to measure all the parameters due to platform limitations. Therefore, the models should include the necessary parameters to allow relevant and adequate predictions.

This thesis work is performed in the context of a European project called BETSY. The BETSY project investigates theory, models and design methodology to make well-founded trade-offs between time-constraints, terminal and network resources and energy consumption. Figure (1) shows a basic BETSY set-up of a video streamed from a camera through a wireless connection to a handheld computer, where it is displayed.

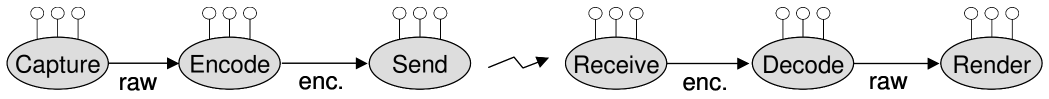


Figure 1: End to End streaming

On the left hand side, the video is captured, encoded (MPEG-4 simple profile) and sent over a wireless link (IEEE 802.11g). On the right hand side, the encoded video is received, decoded and rendered. Each of these functions has a number of parameters that can be set. Some of the parameters are temporal resolution (frames per second), spatial resolution (number of pixels per frame) and bit rate of the incoming stream.

This thesis focuses on developing and experimentally validating power consumption models for the combined decoder (MPEG-4 simple profile) and rendering applications. The application parameters are chosen as the temporal resolution (frames per second) and spatial resolution (number of pixels per frame). In the rest of the thesis, these parameters are referred as frame rate and frame size respectively.

The platform chosen for the experiments is Philips PNX 1500, which mainly consists of a PNX1500 chip (also called SoC in this thesis) and a DDR memory chip.
The platform is chosen such that the power measurements can be performed over different components of the platform separately. It is assumed that for obtaining greater measurement accuracy, we need to separate the component impact parameters properly. With this assumption, we performed the power measurements over SoC and DDR memory separately.

1.2 Energy vs. power consumption

Although, the words energy consumption and power consumption are often used interchangeably, there is an important difference between these two words. The power used by a device is the energy consumed per time unit. Conversely, energy consumption is the time integral of power. In handheld devices a battery stores a given quantity of energy. Therefore, in handheld devices the goal is to reduce the energy consumption to perform all the necessary tasks satisfactorily. Even though minimizing power consumption cannot minimize the energy consumption in all cases, there are some cases where it works. For example, for fixed duration tasks such as playing video or audio, energy consumption is directly proportional to the average power consumption (since the duration of the task is constant). Hence, in this thesis the average power consumed for a fixed duration of task execution is considered.

1.3 Approach

In this thesis, power models for the SoC and memory are developed separately and then these models are combined to get an integrated power model. With reference to the goal mentioned in the previous section, we opted for two methods which are described in the following paragraphs. Both the methods are based on physical measurements in order to guarantee real values with good accuracy.

1.3.1 Black box approach

In this approach, the models are developed at a higher abstraction level by excluding the architecture level details. Therefore, the SoC and memory are considered as a black box. The average power consumption across SoC/memory is measured for different settings of the application parameters: frame rate and frame size. The models are developed by regression on the measurement data. The regression models are improved by making trade-offs between accuracy and simplicity of the models.

1.3.2 White box approach

In white box approach, architectural details of the platform are captured as platform parameters. The white box approach consists of two steps. In the first step, power consumption is expressed as a function of platform parameters. In the second step, the platform parameters are expressed as a function of application parameters to get a high level abstract model that predicts the power consumption from the application parameters. These two steps are explained in detail in the following paragraphs.

In the first step, the power models for the SoC/memory are developed by analysing different states and activities of SoC/memory that cause power consumption. These power models are expressed as a function of different platform parameters. The platform parameters are identified as two types: application independent platform parameters and application dependent platform parameters. The application independent platform parameters are assumed to be independent of the application parameters, frame rate and frame size. The examples of the application independent platform parameters are: average power consumption in different states and activities of the SoC/memory. The application dependent platform parameters are assumed to be dependent on the application parameters and the examples of these parameters are: time spent by SoC/memory in a particular state or activity. The average power consumption and the time spent in different states and activities (application dependent platform parameters) of the SoC/memory are measured for different settings of the application parameters, frame rate and frame size. The average power consumption in different states and activities (application independent platform parameters) is obtained by regression on the measurement data.

In the second step, the time spent in different states and activities (application dependent platform parameters) are related to the application parameters. The models in the second step are also developed by regression on the measurement data. Finally, a compositional model, which replaces the application dependent platform parameters of the SoC/memory models with the second step models, is presented.

1.4 Thesis Overview

This thesis is organized as follows. Chapter 2 introduces the Philips PNX 1500 platform and the tools that are used for performance measurements. Chapter 3 describes the experimental set up and the power measurements. In chapter 4, the basic white box approach power models for the CPU and memory are developed and discussed. Chapter 5 deals with developing power models for the whole SoC by taking the CPU power models presented in Chapter 4 as a reference as well as developing black box approach models for the SoC. Chapter 6 deals with validating the memory models presented in Chapter 4 through experiments as well as developing black box approach models for the memory. In Chapter 7, the SoC and memory black box approach models are combined to get an integrated power model. Chapter 8 presents the experiments performed with a different input stream and discusses the dependency of the average power consumption on the input stream content. Chapter 9 discusses the effect of frequency and voltage scaling on the average power consumption. Chapter 10 summarizes the thesis together with the ideas for future work.

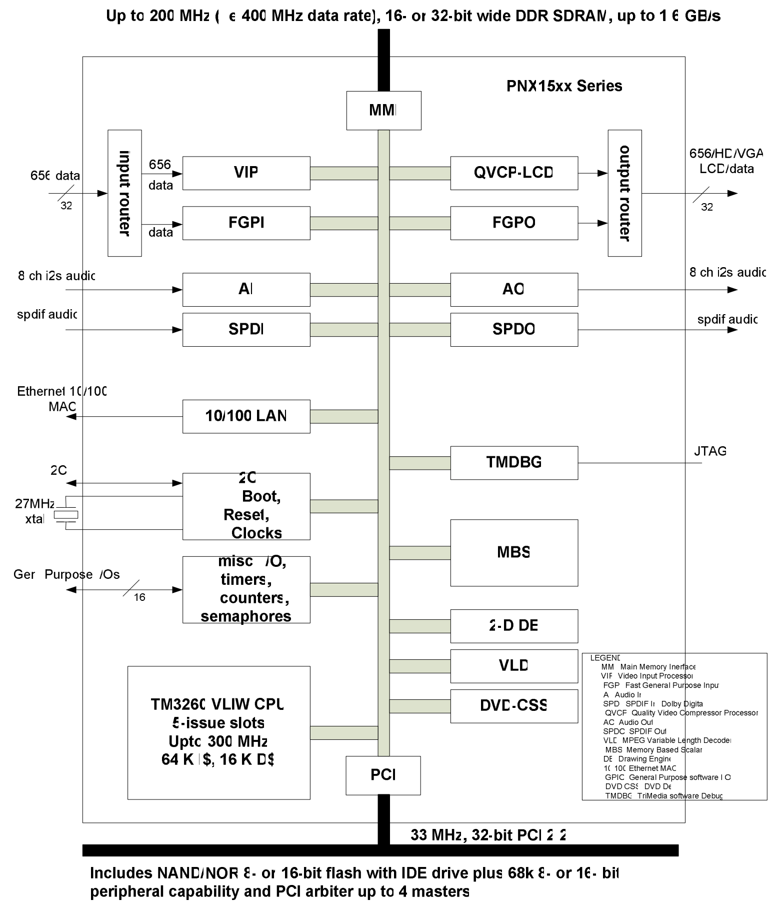
2 Philips PNX1500 and Tools

2.1 PNX1500

The PNX1500 [3] is a complete Audio/Video/Graphics System on Chip. It has a high performance 32-bit VLIW processor, TriMedia TM3260 that can perform high quality audio and video signal processing and can also serve as general-purpose control processor. It runs PSOS operating system. Several image and video processing accelerators in the SoC assist CPU by providing image scaling and composition.

Figure (2) depicts the functional block diagram of PNX1500.The functionality provided by SoC can be divided into three categories: decoding, processing and displaying. MPEG-4 decoding function is implemented in software. Processing and displaying functions are implemented in hardware accelerators. Quality Video Composition Processor (QVCP) provides a high-resolution graphics controller with graphics and video processing. QVCP allows composition of 2 layers, and can output in 656/HD/VGA or LCD format, up to 10-bit per component and up to 81Mpixel/s. Memory Based Scalar (MBS) provides functions like image scaling, video format conversions including colour space conversion, luminance histogram measurements and non-motion/motion compensated de-interlacing. MMI (Main Memory Interface) provides interface between 32-bit, 200MHz, 256MB DDR SDRAM and TM3260 CPU, DMA devices and other internal resources that require memory access. The 32-bit VLIW processor has 5-issue slots, 128 32-bit registers and 16KB data and 64KB instruction cache. Both instruction and data cache are eight-way-set associative and with 64B block size. The TM3260 CPU contains four programmable timer/counters, all with the same function. Three of them are intended for general use where as fourth timer/counter is reserved for use by the system software and should not be used by applications.

The PNX1500 is designed to work in two modes: *standalone* mode and *host* mode [3]. In *standalone* mode, the PNX1500 acts as a master. In this mode, the software application that runs on TM3260 CPU is retrieved from EEPROM or flash memory device. In *host* mode, the PNX1500 acts as a slave. In this mode, the software application is downloaded into PNX1500 main memory (DDR memory) before TM3260 CPU is released from reset. Throughout this project, PNX1500 is used in *host* mode, where it is installed in the PCI slot of the PC. Advantage of PCI interface is fast access to shared memory for download and debug.


Up to 200 MHz (400 MHz data rate), 16- or 32-bit wide DDR SDRAM, up to 1.6 GB/s
PNX15xx Series
MMI
input router
output router
VIP
FGPI
QVCP-LCD
FGPO
AI
AO
SPDI
SPDO
10/100 LAN
TMDBG
MBS
2-D DE
VLD
DVD-CSS
PCI
TM3260 VLIW CPU 5-issue slots Upto 300 MHz 64 K I$, 16 K D$
misc I/O, timers, counters, semaphores
I2C Boot, Reset, Clocks
656 data
8 ch i2s audio
spdif audio
Ethernet 10/100 MAC
I2C
27MHz xtal
Gen Purpose I/Os
656/HD/VGA LCD/data
JTAG
33 MHz, 32-bit PCI 2.2
Includes NAND/NOR 8- or 16-bit flash with IDE drive plus 68k 8- or 16- bit peripheral capability and PCI arbiter up to 4 masters
LEGEND
Figure 2: Functional block diagram of PNX1500 SoC

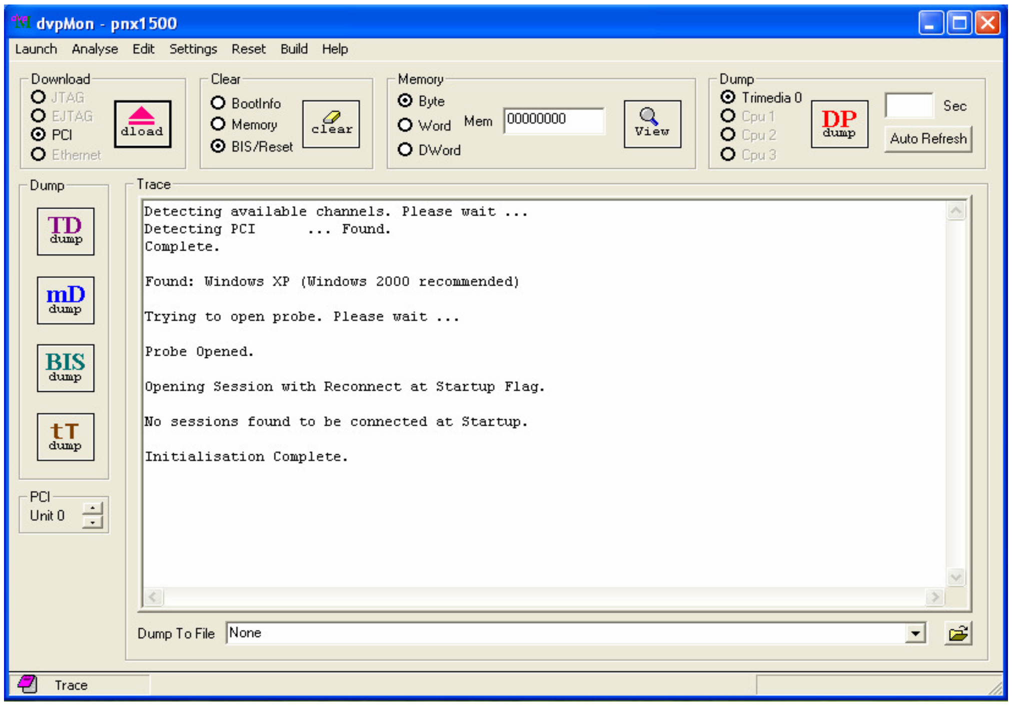
2.2 Tools

Philips provides a few tools [4] [5] along with the NDK distribution to provide a means to interface with the target. These tools provide basic operations like download, execute, basic tracing along with some run time analysis and performance measurements. The following subsections explain the tools that are used to download the application on target architecture and profiling tools that are used for analysis.

2.2.1 dvpMon

dvpMon is a stand-alone Win32 executable that provides a graphical user interface to handle downloads to TriMedia over various channels such as PCI, JTAG, EJTAG, and ETHERNET. Figure (3) is a screenshot of the dvpMon. dvpMon has the following features:

- Download files in .elf, .bin, .mi and .out format
- Start and Reset TriMedia
- Dump traces from TimeDoctor, Memory
- View memory
- View and update BIS (Boot Info Structure)
- Look up DVP error codes
- Launch other tools like TimeDoctor viewer and URD



```
Detecting available channels. Please wait ...
Detecting PCI      ... Found.
Complete.

Found: Windows XP (Windows 2000 recommended)

Trying to open probe. Please wait ...

Probe Opened.

Opening Session with Reconnect at Startup Flag.

No sessions found to be connected at Startup.

Initialisation Complete.
```

Download: JTAG EJTAG [✓ PCI] Ethernet
Clear: BootInfo Memory [✓ BIS/Reset]
Memory: [✓ Byte] Word DWord
Dump: [✓ Trimedia 0] Cpu 1 Cpu 2 Cpu 3

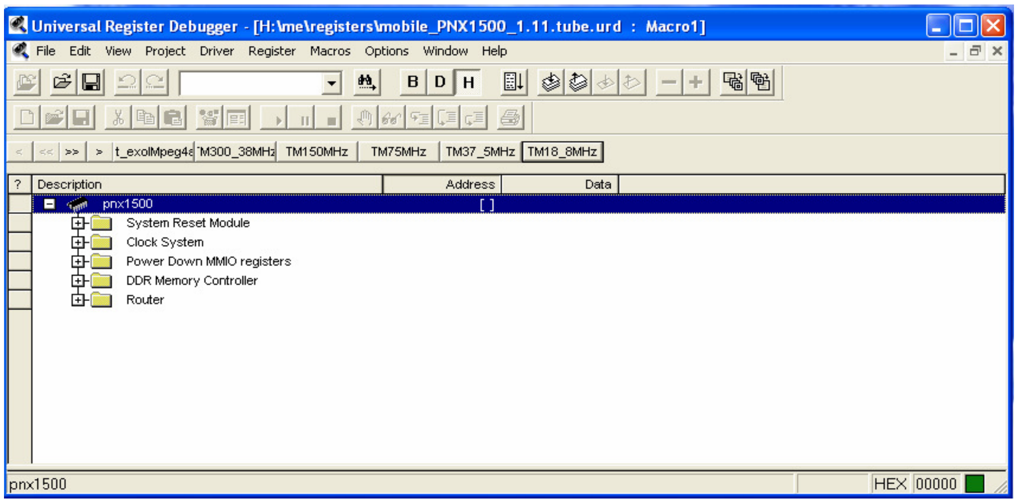
Figure 3: Screenshot of dvpMon

2.2.2 URD (Universal Register Debugger)

URD is used to debug registers on target, the PNX1500. URD environment consists of an application core and a set of supporting files. The application core provides the basic URD functionality and the support files customize the URD to access and manipulate the target device. The PC, which runs the URD application and the target, the PNX1500 are communicated through PCI.

Device Description files (*.URD) describe the registers of the target hardware. Using these register settings, it is possible to change the frequency of CPU and other hardware blocks on SoC. Current register values can be stored in *Current Register Settings* (*.URG) files. Sequences of register accesses can be described with URD basic macros and stored in a *URD Basic Macro description file* (*.URM). These files are used together with the corresponding *Device Description file*.

Figure (4) shows the screen shot of URD, in which a .urd file gives information about the registers of the PNX1500 target namely System Reset Module, Clock System, Power Down MMIO registers, DDR memory controller and Router. It is possible to read and write the register values on target. It is also possible to reset the target using URD.


Universal Register Debugger - [H:\me\registers\mobile_PNX1500_1.11.tube.urd : Macro1]
File Edit View Project Driver Register Macros Options Window Help
B D H
t_exolMpeg4 M300_38MHz TM150MHz TM75MHz TM37_5MHz TM18_8MHz
Description Address Data
pnx1500 [ ]
System Reset Module
Clock System
Power Down MMIO registers
DDR Memory Controller
Router
pnx1500 HEX 00000
Figure 4: Screen shot of URD

2.2.3 TimeDoctor

TimeDoctor is a profiling tool that allows users to visualize and analyse TriMedia programs. In order to do profiling using TimeDoctor tool, it is necessary to compile the entire platform and application with the TimeDoctor build options and to call some initialization functions in the application.

TimeDoctor provides profiling information about Task CPU usage, ISR CPU usage, User Block CPU usage, Cache events, Queues, Semaphores and System events. This profiling data is obtained by instrumenting OSAL functions using the callout facility of the OSAL, and by instrumenting the PSOS task switch. Users can also call the TimeDoctor API directly to define the user events. Time Doctor adds a small amount of overhead on the system because it calls the callout functions for all the OS events [4]. This data can then be collected, filtered and formatted. Graphical output traces will be written to an ASCII .tdi file for importing into the TimeDoctor Viewer.

TimeDoctor provides profile information in three phases namely data collection, data processing and data presentation. TimeDoctor viewer is used to display data generated by TimeDoctor in graphical format. Figure (5) is a screenshot of TimeDoctor viewer.

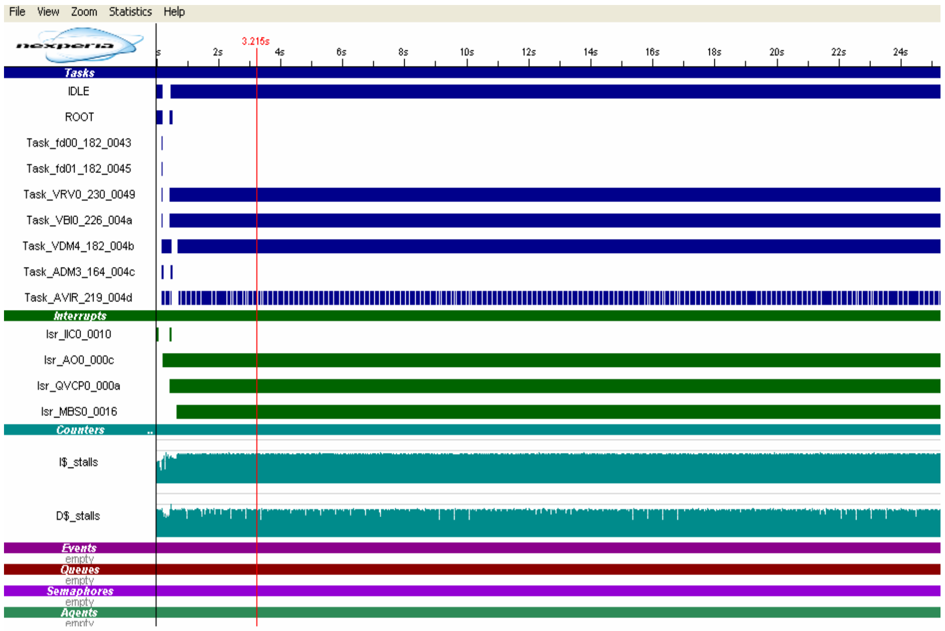


Figure 5: Screenshot of Time Doctor Viewer

3 Experimental setup and Measurements

3.1 Introduction

The previous chapter discussed about the Philips PNX 1500 platform and the tools that are used for performance measurements. This chapter describes the experimental setup and the power measurements. In this chapter, we also discuss the possible sources of errors in measurements and modelling.

3.2 Experimental Setup

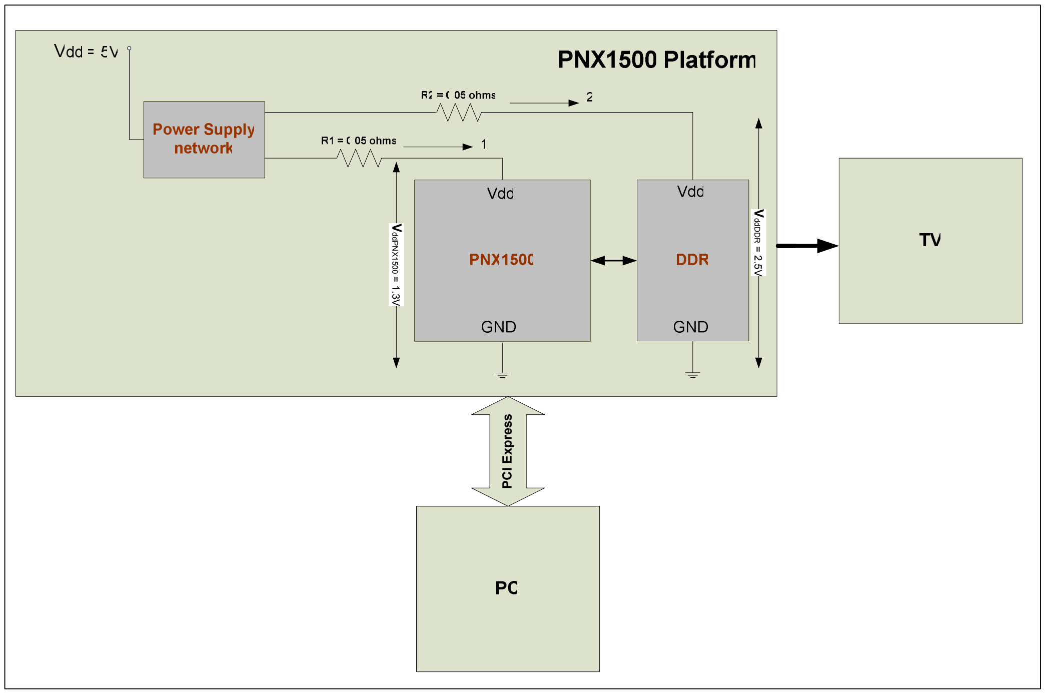


Figure 6: Experimental set-up

Figure (6) shows the experimental set-up block diagram. This block diagram consists of three blocks namely PC, PNX1500 platform and TV. PCI express cable connects the PC with the PNX platform board. This connection is used to download the built-in application from the PC onto the PNX1500 board. Output from displaying functions of the PNX1500 board is connected to TV through a cable.

Only the relevant components to this thesis are shown in the PNX platform board. The supply voltage for the PNX1500 chip (also called SoC in this thesis) and DDR memory chip are derived from the power supply network. The resistors R1 and R2 are also part of this power supply network. These resistors are shown external to the power supply network to understand the current flow into the PNX1500 chip and memory chip. The supply voltage to SoC is called $\mathbf{V}_{\text{ddPNX1500}}$ and its value is 1.3 V. Similarly, the supply voltage to DDR memory is called $\mathbf{V}_{\text{ddDDR}}$ and its value is 2.5 V.

3.3 Power measurements

The average power consumption of SoC/Memory is the product of voltage across SoC/Memory and current drawn by SoC/Memory. Since R1 is in series with the SoC and R2 is in series with memory, the current drawn by SoC is I1 and the current drawn by memory is I2. Therefore, the power consumption equations can be written as follows:

$P_{SOC} = V_{ddPNX1500} \times I1$
$P_{mem} = V_{ddDDR} \times I2$

The currents I1 and I2 are calculated as follows:

I1 = (Voltage across R1) / R1 = V1 / R1
I2 = (Voltage across R2) / R2 = V2 / R2

Then, the power consumption equations become

$P_{SOC} = (V_{ddPNX1500} \times V1) / R1$
$P_{mem} = (V_{ddDDR} \times V2) / R2$

V_1 and V_2 are the average voltages measured across the resistors R_1 and R_2 respectively. Voltage across the resistors is measured using Keithly Model 2700 Multimeter / Data acquisition system [6]. The voltage measured using the instrument is the average voltage, which is averaged over multiple samples. Therefore, random error (refer Section 3.4) is averaged over multiple samples. The instrument is set to display up to two decimal digits and has a precision of 0.01mV. All the measurements in this thesis were taken at the same offset power which is obtained by resetting the board before starting the measurements. This avoids errors in the measurements.

3.4 Errors

An error is defined as the difference between the measured value and the true value. The sources of errors in measurement and modelling of this work are divided into two types. One is the measurement error and the other one is the modelling error. The following paragraphs describe these errors.

- Measurement errors: The measurement errors are of two types: Random error and Systematic error. The random error is caused by any factors that randomly affect the measurement of the variable across the sample. The important thing about random error is that it does not have any consistent effects across the entire sample, instead it pushes the observed scores up or down randomly. This means that if we could see all of the random errors in a distribution they might add up to zero. Systematic error is caused by any factors that systematically affect measurement of the variable across the sample. Systematic errors are caused by the flaw in the measurement instrument or flaw in the method of selecting a sample or flaw in the technique of estimating a parameter or can be due to inappropriate assumptions about formulae. To minimise the systematic errors, it is necessary to check the instrument and assumptions continuously. The sources of measurement errors in this thesis are the power measuring instrument, Time Doctor tool and the assumptions in the models.

- Modelling errors: Modelling error depends on how well the assumed model suits the data. Root Mean Square Error is a measure for the accuracy of the models because it is measured in the same units of data and is a representative of the size of a typical error. Two models whose RMSEs are in the same units can be compared to see which one is more accurate. Another important parameter to be considered in comparing the models is the complexity of the model. When we trade off model complexity against error measures, it is possibly not worth adding another independent parameter to a regression model to decrease the RMSE by only a few more percent. Therefore, when the RMSE of two models is not deviating much then it is better to choose the model with less number of parameters.

4 Energy and Power models for the CPU and Memory

4.1 Introduction

In order to develop power models for the SoC/memory using white box approach, we need to analyse the sources of power consumption in these hardware components. The sources of power consumption in SoC are the CPU and hardware accelerators of the SoC. This chapter discusses the basic concepts of the power consumption in a CMOS circuit and the power consumption in an embedded processor and memory considering fixed and variable frequency and voltage methods. After detailed analysis of various power consuming states and activities of the CPU/memory, we present power models for the CPU/memory in terms of the platform parameters. Next chapter develops the power models for the whole SoC by adding the influence of hardware accelerators to the CPU power model presented in this chapter.

4.2 Power consumption of a CMOS circuit

The power consumption of any CMOS circuit is expressed as the sum of switching power, leakage power and short-circuit power [7]. Switching power is caused by the switching activity (charging and discharging) of the capacitor. A portion of the power is consumed during the switching activity due to the short circuit at the driving gate's output, which is referred as short-circuit power. Switching and short-circuit powers form the dynamic power consumption. There is also a portion of power consumed irrespective of the switching activity, which is referred as leakage power or static power consumption. Typically, the short-circuit power is a small percentage, less than 10% of the total power consumption; ignoring short-circuit power results in the following average power consumption equation [7].

$$P = P_{switch} + P_{leakage} \approx C \times V^2_{DD} f + V_{DD} \times I_{leakage} \qquad (1)$$

In the above equation, C is a constant representing the average capacitance resulting from all the active switching cells, V_{DD} is the supply voltage, f is the clock frequency, and $I_{leakage}$ is the average leakage current.

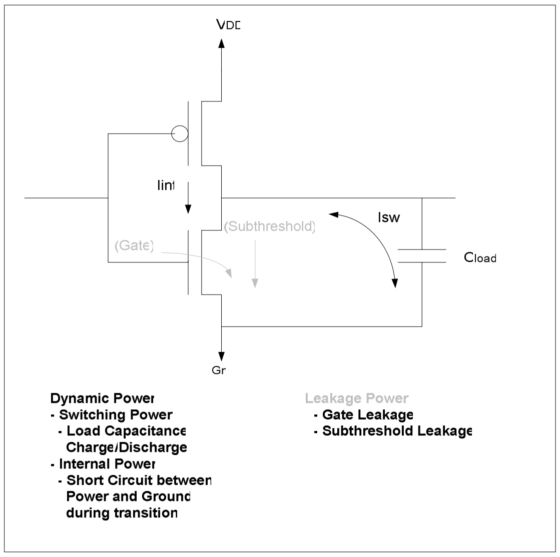


Figure 7: Power dissipation in CMOS designs

4.3 Energy consumption of an embedded processor and memory

In an embedded system, performing a given task with a given time constraint can be achieved in different ways [7]. One of them is the fixed frequency and voltage scheme. The other one is variable frequency and voltage scheme.

In case of fixed frequency and voltage scheme, the processor and memory are designed to operate at a supply voltage and frequency that satisfies the timing constraints for the worst-case scenario. When a low timing constraint task has to be executed, then even after finishing the task the processor and memory consume power. For example, for a decoder application depending on the frame rate of the input stream, processor and memory are in idle between the frames. During these idle periods, the processor and memory consume considerable amount of power.

In case of variable frequency and voltage scheme, the operating frequency is scaled according to the timing constraints of the application. Processor could lower the frequency for a low timing constraint task and can increase the frequency for a high timing constraint task.

Consider a fixed duration task of period T. For fixed frequency and voltage scheme average energy consumption for a given task completed in time T1 < T is given by:

$$\begin{aligned} E_{\mathrm{FIXED}} &= \int_0^T (P_{switch} + P_{leakage})dt \\ &\approx \int_0^{T1} (C_1 \times V_{DD}^2 f)dt + \int_{T1}^{T} (C_2 \times V_{DD}^2 f)dt + \int_0^T (V_{DD} \times I_{leakage})dt \end{aligned} \qquad (2)$$

Where C_1 is the average switching capacity during task processing and C_2 is the average switching capacity after the task is completed. Operating the processor and memory in standby (clock shutdown) state after the task is completed at T1, saves the switching power. Then the Equation (2) becomes:

$$E_{\mathrm{fixed}} = \int_0^T (P_{switch} + P_{leakage})dt \approx \int_0^{T1} (C_1 \times V_{DD}^2 f)dt + \int_0^T (V_{DD} \times I_{leakage})dt \qquad (3)$$

For a variable frequency and voltage scheme, the clock frequency *f1* is reduced such that the same task can be completed in time T. Accordingly, the supply voltage is changed to V_{DD1} for the reduced frequency *f1*. In this case the average energy consumption is given by:

$$E_{\mathrm{variable}} = \int_0^T (P_{switch} + P_{leakage})dt \approx \int_0^T (C_1 \times V_{DD1}^2 f_1)dt + \int_0^T (V_{DD1} \times I_{leakage})dt \qquad (4)$$

4.4 Energy and Power consumption models for CPU and Memory

Three forms of energy consumption are identified for any hardware block: static frequency dependent and activity dependent energy consumption [8] [9]. The last two contribute to the dynamic energy consumption of the hardware block. The static power consumption depends on the state and voltage of the hardware block. Frequency dependent power consumption depends on state, voltage and the clock frequency at which the hardware block is operating. Activity dependent power consumption depends on the state, voltage and the frequency of occurrence of an activity in the given time interval. In some cases, activity dependent power consumption can become the frequency dependent power consumption. For example when each clock cycle is viewed as an activity then the activity dependent power consumption is the same as frequency dependent power consumption.

Total power consumption in an interval T is obtained by summing up the three forms of power consumptions over all the states, voltages, frequencies and activities. In case of fixed frequency and voltage scheme, the summation is only over states and activities. Since only fixed frequency and voltage scheme is considered in this thesis, the power consumption models for CPU and memory will be described for this scheme only.

4.4.1 Energy and Power consumption models for CPU

This section presents energy and power consumption models for a CPU with cache. These models are developed by identifying different states and activities of CPU that cause power consumption. The power consumption of the CPU in one state is different from the power consumption in another state. Similarly, power consumption of CPU for one activity is different from the power consumption of another activity.

For a CPU with cache, three different states are identified: *active, stall and idle* states [8]. CPU is in active state when it actually computes. In active state all the CPU's logic is connected to the clock.

In modern CPUs, most of the memory accesses are to the cache. Memory accesses are described as a read or write to the cache. An access to the cache is called cache hit when a read or a write is succeeded, i.e. the block requested is available in cache in case of read and a block can be written in to a particular location in case of a write. A cache read miss occurs when the block is not available in the cache and has to be fetched from the external memory. Cache write miss occurs when a write to a particular location is not possible because it is not empty. In case of write back, write allocate caches, the block in the required location is written back to the main memory if that block is dirty and then the requested block is loaded into that location. In case of write through and write no allocate caches; the requested block is updated in main memory only. Upon a cache miss CPU enters into the stall state. In stall state some part of the CPU's logic is disconnected from the clock.

CPU is in idle state when there is no task to be performed. In idle state, large part of the CPU's logic is disconnected from the clock.

It is assumed that there is no other activity dependent power consumption needs to be identified since the total power consumption of the CPU is captured by these three states. With this assumption, the average power consumption of the CPU is expressed as the sum of the power consumptions in individual states.

$$P_{CPU,T} = P_{CPU,active} \times t_{CPU,active} + P_{CPU,stall} \times t_{CPU,stall} + P_{CPU,idle} \times t_{CPU,idle} \quad (5)$$

Energy consumption of CPU during period T, is expressed as the sum of energy consumption in individual states.

$$E_{CPU,T} = P_{CPU,active} \times T_{CPU,active} + P_{CPU,stall} \times T_{CPU,stall} + P_{CPU,idle} \times T_{CPU,idle} \quad (6)$$

$P_{CPU,T}$: Average power consumption of CPU during interval T
$E_{CPU,T}$: Energy consumption of CPU during interval T
$P_{CPU,active}$: Average power consumption of CPU in active state
$P_{CPU,stall}$: Average power consumption of CPU in stall state
$P_{CPU,idle}$: Average power consumption of CPU in idle state
$t_{CPU,active}$: Fraction of time CPU is in active state
$t_{CPU,stall}$: Fraction of time CPU is in stall state
$t_{CPU,idle}$: Fraction of time CPU is in idle state
$T_{CPU,active}$: Time spent by CPU in active state
$T_{CPU,stall}$: Time spent by CPU in stall state
$T_{CPU,idle}$: Time spent by CPU in idle state

The power consumption in each state is the sum of static and frequency dependent power consumption, since there is no activity dependent power consumption.

$$P_{CPU,active} \approx V_{DD} \times I_{leakage} + C1 \times V^2_{DD} f$$
$$P_{CPU,stall} \approx V_{DD} \times I_{leakage} + C2 \times V^2_{DD} f$$
$$P_{CPU,idle} \approx V_{DD} \times I_{leakage} + C3 \times V^2_{DD} f$$

Since we are considering only fixed frequency and voltage scheme, the power consumption in each state is a fixed constant value at a particular frequency and voltage. Therefore the model for predicting the average power consumption of the CPU (refer Equation (5)) is expressed as the linear sum of average power consumption in individual states.

4.4.2 Energy and Power consumption models for Memory

This section presents the energy and power consumption models for a DDR memory. Similar to CPU models, the memory models are developed by identifying various power consuming states and activities of memory [8] [9].

DDR memory stands for Double Data Rate memory, which means two data transfers take place per clock cycle. Dynamic memory must be refreshed regularly, with a given maximum refresh interval for each page in each bank.

In dynamic memory the data transfers are not with the memory itself, but with sense amplifiers. Before reading or writing, the contents of one page in one bank are loaded to sense amplifiers. The act of loading to sense amplifiers is called as activation.

Activation destroys the data in the memory bank. Therefore, it is necessary to restore the page in the bank. The act of restoring is called as precharge.

Two different states of the DDR memory are identified: *active* and *idle* states. DDR memory is in active state when at least one page is activated. DDR memory enters into idle state when all the pages are precharged.

A read or write burst is a sequence of bytes read from or written to the same page of same bank without interruption. Therefore, the burst is viewed as a sequence of words belonging to the same page in the same bank. Every burst is preceded by an activation of the page and followed by a precharge.

We assumed that reads and writes can take place in *active* state only. The cost of reading or writing a word is captured by the activities *read* and *write.* The energy cost due to activation and precharge is captured by the activity *burst.*

Time spent in the *active* state, but not used for data transfer is known as *stall* time. Stall time includes refresh time, and also includes transition costs of different types. To capture these transitions, we use the notion of *efficiency*, which captures transfer time as a fraction of the total active time. Efficiency is used to calculate the total active time of the DDR memory using the following formula:

$$t_{mem,active,T} = (n_{mem,read,T} + n_{mem,write,T}) / (f_{mem} \times eff_{mem}) \qquad (7)$$

With these assumptions and definitions the following models for the energy and power consumption of the memory is developed.

T	: Length of the total time interval
$E_{mem,T}$	: Enrgy consumption of memory during interval T
$P_{mem,T}$	: Average power consumption of memory during interval T
$P_{mem,active}$	: Power consumption of the memory during active state
$P_{mem,idle}$	: Power conmsumption of the memory during idle state
$e_{mem,read}$	: Energy cost of one read
$e_{mem,write}$	: Energy cost of one write
$e_{mem,burst}$	: Energy cost of one burst
$t_{mem,active}$	: Fraction of time memory is in active state
$t_{mem,idle}$	: Fraction of time memory is in idle state
$T_{mem,active}$	: Time spent by memory in active state
$T_{mem,idle}$	: Time spent by memory in idle state
$n_{mem,read,T}$	: Number of occurances of activity read during interval T
$n_{mem,write,T}$	: Number of occurances of activity write during interval T
$n_{mem,burst,T}$	: Number of occurances of activity burst during interval T
$f_{mem,read}$	: Frequency of the read activity
$f_{mem,write}$	: Frequency of the write activity
$f_{mem,burst}$	: Frequency of the burst activity

$$E_{mem,T} = P_{mem,active} \times T_{mem,active} + P_{mem,idle} \times T_{mem,idle} + n_{mem,read,T} \times e_{mem,read} + n_{mem,write,T} \times e_{mem,write} + n_{mem,burst,T} \times e_{mem,burst} \qquad (8)$$

$$P_{mem,T} = P_{mem,active} \times t_{mem,active} + P_{mem,idle} \times t_{mem,idle} + f_{mem,read} \times e_{mem,read} + f_{mem,write} \times e_{mem,write} + f_{mem,burst} \times e_{mem,burst} \quad (9)$$

Above equation is represented as follows for the convenience of notation:

$$P_{mem,T} = P_{mem,active} \times t_{mem,active} + P_{mem,idle} \times t_{mem,idle} + P_{mem,read} \times t_{mem,read} + P_{mem,write} \times t_{mem,write} + P_{mem,burst} \times t_{mem,burst} \quad (10)$$

In the above equation $f_{mem,read} \times e_{mem,read}$ term of Equation (9) is replaced with $P_{mem,read} \times t_{mem,read}$, both the terms give the average power consumption of the read activity. Similarly, the terms $f_{mem,write} \times e_{mem,write}$ and $f_{mem,burst} \times e_{mem,burst}$ are replaced with the terms $P_{mem,write} \times t_{mem,write}$ and $P_{mem,burst} \times t_{mem,burst}$ respectively.

Figure (8) represents the memory power model given in Equation (10). The *burst* activity overlaps in time with *read* and *write* activities because of the multiple banks in DDR memory. Usually, the transition costs between *read* and *write* activities are included into *write* activity. Therefore, in Figure (8), the energy cost of *write* activity is more than that of *read* activity.

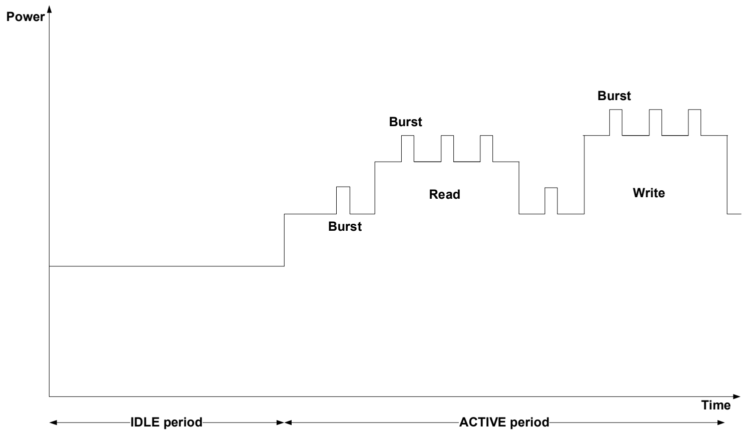


Figure 8: Representation of memory power model given in Equation (10)

In later chapters, for validating the memory power model (Equation (10)) we measure the parameters of the model. Parameters $t_{mem,idle}$, $t_{mem,read}$ and $t_{mem,write}$ of the model can be measured through performance counters of DDR controller. But, we found that, DDR controller of this platform has no counters to measure the $t_{mem,burst}$ of the memory model. Therefore, with the measurable parameters Equation (10) is modified as follows:

$$P_{mem,T} = P_{mem,active} \times t_{mem,active} + P_{mem,idle} \times t_{mem,idle} + P_{mem,read} \times t_{mem,read} + P_{mem,write} \times t_{mem,write} \quad (11)$$

Here,we can measure $t_{mem,idle}$ directly through performance counters. Therefore $t_{mem,active}$ is calculated as follows instead of calculating using efficiency (Equation (7)):

$$t_{mem,active} = 1 - t_{mem,idle}$$

The following figure represents the modified memory power model in Equation (11).

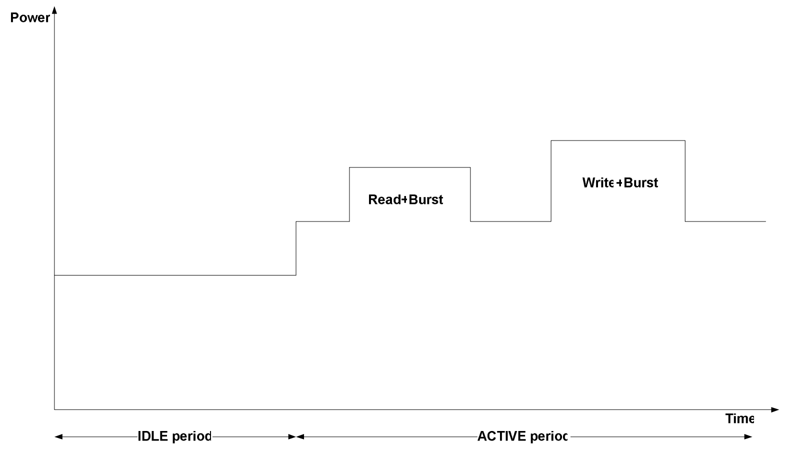


Figure 9: Representation of modified memory power model given in Equation (11)

In the modified model, we are not considering the average power consumption due to *burst* activity separately. Therefore, the average power consumption due to *burst* activity is included into the average power consumption of activities *read* and *write*.

4.4.3 Parameters of the CPU and memory power models

The parameters of the CPU and memory power models (refer Equation (5) and (10)) are divided into two sets:

1. Application independent platform parameters : $P_{CPU,active}$, $P_{CPU,stall}$, $P_{CPU,idle,}$ $P_{mem,active}$, $P_{mem,idle}$, $P_{mem,read}$, $P_{mem,write}$

2. Application dependent platform parameters: $t_{CPU,active}$, $t_{CPU,stall}$, $t_{CPU,idle}$, $t_{mem,active}$, $t_{mem,read}$, $t_{mem,write}$

Application independent platform parameters are assumed to be specific for the CPU/memory and its settings, but are independent of the specific context in which the CPU/memory is being used. For example, power consumed by CPU/memory in a particular state or for a particular activity depends on the amount of logic that is active during these states and activities and is expected to be specific for a CPU/memory irrespective of the application. With this hypothesis, it is assumed that the application independent platform parameters are independent of the application parameters.

Application dependent platform parameters are assumed to be dependent on the application parameter settings. For example, if there are more number of frames (frame rate) or more number of pixels per frame (frame size) to be processed by CPU, then we can expect that the CPU spends more time in active state and less time in idle state. Similarly, we can expect more number of accesses to memory in this scenario. With this hypothesis, it is assumed that the application dependent platform parameters are dependent on the application parameters.

5 Power Models for System-on-Chip (SoC)

5.1 Introduction

In the previous chapter, we have developed the white box model for the CPU and memory considering different power consuming states and activities of the CPU and memory. This chapter develops power model for the whole SoC by taking the CPU power model (refer Equation (5)) as a reference. This is achieved by adding the influence of other hardware components of the SoC to the CPU power model step by step through experiments.

The white box approach consists of two steps:

(1) Application dependent platform parameters ($t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$) and the average power consumption across SoC ($P_{SoC,T}$) are measured experimentally for the given frame rate and frame size of the input stream. Three different frame rates (30fps, 25fps and 12.5fps) and frame sizes (4cif, cif and qcif) are considered in the experiments. Application independent platform parameters ($P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$) are calculated by performing linear regression [10] on the equations substituted with the experimentally measured values for application dependent platform parameters and $P_{SoC,T.}$

(2) The application dependent platform parameters of the SoC power model are expressed as a function of application parameters (frame rate and frame size) through regression models.

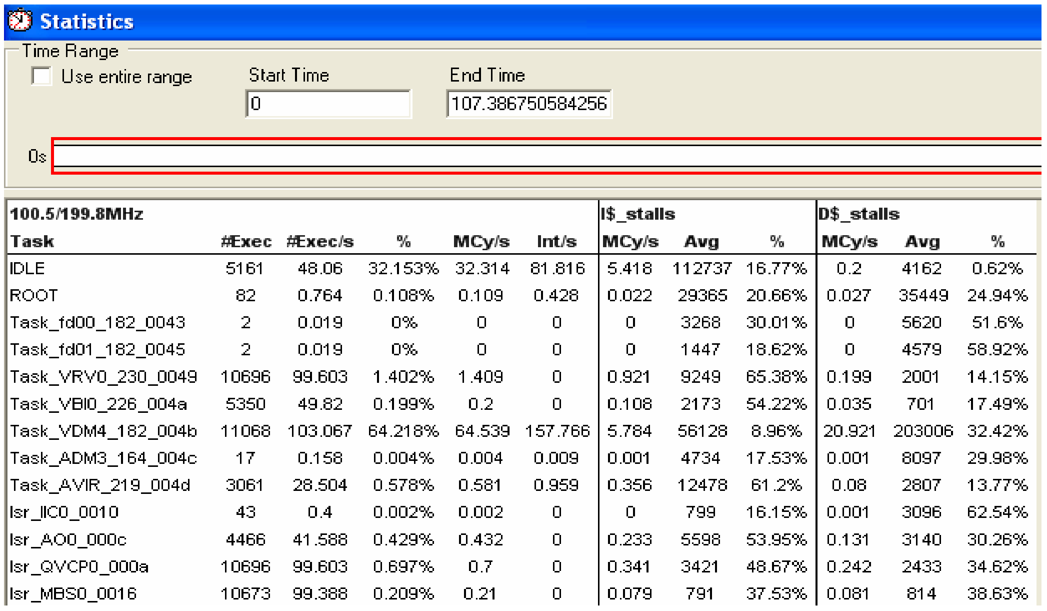
Finally, from the models of each step described above, a compositional model for the power consumption of the SoC in terms of applcation parameters is developed. Using the compositional model, we can predict the average power consumption of the SoC for any values of frame rate and frame size.

This chapter also develops the black box models without considering the architecture level details.

5.2 White box approach experiments and results

The Average power consumption measured across the SoC ($P_{SoC,T}$) during the execution of the decoder application, not only consists of power consumption due to CPU, but also power consumption due to other hardware blocks. A fundamental aspect of the PNX15xx Series system is to provide hardware modules (or hardware accelerators) that relieve the TM3260 CPU for other video/audio processing [3]. That means CPU and hardware blocks work simultaneously.

The application dependent platform parameters ($t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$) are measured using TimeDoctor tool. Figure (10) is a screen shot of the TimeDoctor statistics. All tasks except IDLE and ROOT are dynamically created at runtime by providing system calls to the PSOS kernel [11]. The purpose of IDLE task is to consume CPU cycles when no other task is running. Statistics in Figure (10) give the number of execution cycles of each task and how much percentage of execution cycles are stall cycles. The active and stall cycles of the CPU are calculated as the sum of the individual task cycles.


Statistics

Time Range

Use entire range — Start Time: 0 — End Time: 107.386750584256

0s

100.5/199.8MHz						I$_stalls			D$_stalls		
Task	**#Exec**	**#Exec/s**	**%**	**MCy/s**	**Int/s**	**MCy/s**	**Avg**	**%**	**MCy/s**	**Avg**	**%**
IDLE	5161	48.06	32.153%	32.314	81.816	5.418	112737	16.77%	0.2	4162	0.62%
ROOT	82	0.764	0.108%	0.109	0.428	0.022	29365	20.66%	0.027	35449	24.94%
Task_fd00_182_0043	2	0.019	0%	0	0	0	3268	30.01%	0	5620	51.6%
Task_fd01_182_0045	2	0.019	0%	0	0	0	1447	18.62%	0	4579	58.92%
Task_VRV0_230_0049	10696	99.603	1.402%	1.409	0	0.921	9249	65.38%	0.199	2001	14.15%
Task_VBI0_226_004a	5350	49.82	0.199%	0.2	0	0.108	2173	54.22%	0.035	701	17.49%
Task_VDM4_182_004b	11068	103.067	64.218%	64.539	157.766	5.784	56128	8.96%	20.921	203006	32.42%
Task_ADM3_164_004c	17	0.158	0.004%	0.004	0.009	0.001	4734	17.53%	0.001	8097	29.98%
Task_AVIR_219_004d	3061	28.504	0.578%	0.581	0.959	0.356	12478	61.2%	0.08	2807	13.77%
Isr_IIC0_0010	43	0.4	0.002%	0.002	0	0	799	16.15%	0.001	3096	62.54%
Isr_AO0_000c	4466	41.588	0.429%	0.432	0	0.233	5598	53.95%	0.131	3140	30.26%
Isr_QVCP0_000a	10696	99.603	0.697%	0.7	0	0.341	3421	48.67%	0.242	2433	34.62%
Isr_MBS0_0016	10673	99.388	0.209%	0.21	0	0.079	791	37.53%	0.081	814	38.63%

Figure 10: Statistics given by Time Doctor

The application independent platform parameters ($P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$) can not be measured directly. Two approaches were taken to obtain $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle.}$ These approaches are explained in the paragraphs below.

In the first approach, it is assumed that the hardware blocks would be in active state only during CPU active periods. Therefore, $P_{CPU,active}$ includes the power consumption due to the active CPU and hardware blocks. Several experiments were performed with the decoder program using a sample stream from the BETSY project with three different resolutions 4cif,cif and qcif and three different frame rates 12.5fps, 25fps and 30fps. To characterize the power consumption during decoder program when CPU is in stall and idle states, some test programs were executed. In Section 5.2.1 and 5.2.2 we explain these test programs. The values obtained for $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$ during the execution of decoder and test programs are substituted in the following Equation (12). This equation is considered by taking the CPU power model (Equation (5)) of previous chapter as a reference.

$$P_{SoC,T} \;=\; P_{CPU,active} \times t_{CPU,active} + P_{CPU,stall} \times t_{CPU,stall} + P_{CPU,idle} \times t_{CPU,idle} \qquad (12)$$

Linear equations obtained by the decoder program and test programs are solved to get $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle.}$

The second approach for obtaining $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ is to perform linear regression on the available data. Linear regression is performed on the nine linear equaions obtained from the nine different experiments with the decoder program.

For each of the nine different experiments performed on decoder program, the values obtained for $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$ are shown in the Table (1).

In PNX 1500, the maximum frequency at which CPU can operate is 300.375MHz. When the decoder program is run at this frequency with an input stream of 30fps frame rate and 4cif resolution, the CPU spends 78% of the time in idle state. That means most of the time CPU is in idle state. Running CPU at higher frequency with 78% of idle time is not an optimal condition for the power consumption. Therefore in order to get the best optimal condition for the power consumption we chose 100.5MHz frequency such that the time spent by CPU in idle state is around 10%. The 10% of the margin is left to make sure that the CPU meets the timing constraints.

	FR(fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$P_{SoC,T}$ (mW)
1	30	4cif	0.49	0.39	0.12	717.1
2	30	cif	0.16	0.26	0.59	689.3
3	30	qcif	0.07	0.22	0.71	679.2
4	25	4cif	0.26	0.35	0.38	708.8
5	25	cif	0.12	0.24	0.63	683.8
6	25	qcif	0.05	0.22	0.73	676.7
7	12.5	4cif	0.23	0.28	0.49	701.9
8	12.5	cif	0.08	0.22	0.69	684.3
9	12.5	qcif	0.04	0.20	0.76	674.2

Table 1: $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$ values obtained through various experiments

To obtain the power consumption across the SoC when the CPU is in idle and stall states, two test programs idle_test and stall_test were executed.

5.2.1 Power consumption in idle state

During the CPU idle state, PSOS runs an idle task. The purpose of PSOS idle task is to simply consume the CPU cycles when there is no other task to be performed by CPU. PSOS idle task is nothing but an infinite loop [11].

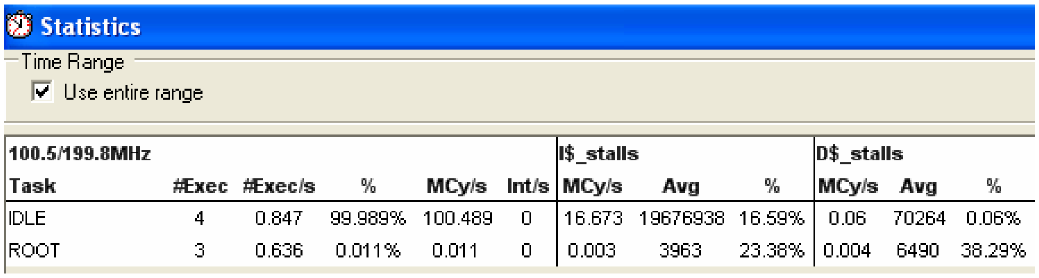
With the assumption that the hardware blocks are active only during CPU active state, the power consumption when CPU is in idle ($P_{CPU,idle}$) would be the clock power of CPU and hardware blocks and the power consumption due to idle task execution.

To get the power consumption across SoC during idle state of CPU ($P_{CPU,idle}$), an idle_test program is executed. During the execution of this program, the hardware blocks that are active during decoder program are clocked. The hardware blocks that are active during decoder program are obtained by using Universal Register Debugger (URD) tool described in Section 2.2.2.

The test program is in C and the *main*() function consists of only *getch*() function. Because of *getch*() function, until a character is entered from the keyboard, the CPU would be in idle state and hence PSOS idle task would be executed. The values measured for $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$ and $P_{SoC,T}$ during this test program are substituted in Equation (12).

$$666.6 = P_{CPU,active} \times 0.00004 + P_{CPU,stall} \times 0.166 + P_{CPU,idle} \times 0.83 \qquad (13)$$

From the Equation (13), it can be seen that CPU spends only 83% of the time in idle state during the execution of idle task. The remaining percentage of time is spent in stall state. The statistics(Figure (11)) show that the 16.6% of stalls during idle task execution are instruction cache stalls. We assume that the instructions of the idle task are flushed out of the cache for some reason and there is a need to get the instructions back from external memory each time the idle task is executed.



Statistics

Time Range

☑ Use entire range

100.5/199.8MHz						I$_stalls			D$_stalls		
Task	#Exec	#Exec/s	%	MCy/s	Int/s	MCy/s	Avg	%	MCy/s	Avg	%
IDLE	4	0.847	99.989%	100.489	0	16.673	19676938	16.59%	0.06	70264	0.06%
ROOT	3	0.636	0.011%	0.011	0	0.003	3963	23.38%	0.004	6490	38.29%

Figure 11: Statistics given by TimeDoctor for idle_test program

5.2.2 Power consumption in stall state

Similarly, with the assumption that the hardware blocks are active only during CPU active state, the power consumption across SoC, when CPU is in stall state would be the stall power of CPU and the clock power of hardware blocks.

To obtain the power consumption across SoC during stall state of CPU, a stall_test program was executed. This test program creates an array in the data cache of the CPU. Each 64th location (each location equals 1B) of the array is read in the program.As the cache line size of the TM3260 is 64B, every cache read of this program creates a miss and brings 64B of data from external memory. Therefore, on every read of this program the CPU stalls for the data from external memory. The values obtained for $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$ and $P_{SoC,T}$ during this test program are substituted in Equation (12).

$$654 = P_{CPU,active} \times 0.06 + P_{CPU,stall} \times 0.93 + P_{CPU,idle} \times 0.004 \qquad (14)$$

5.2.3 Linear equation solutions

Below, we give an example of how the equations are solved to obtain $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ values. Equation(15), is obtained from the decoder program with an input stream of 4cif resolution and 30fps (from Table(1)).

$$717.1 = P_{CPU,active} \times 0.49 + P_{CPU,stall} \times 0.39 + 666.6 \times 0.12 \qquad (15)$$

Equation (15) along with the equations from idle_test and stall_test (Equation (13) and (14) respectively) is written in the following matrix form.

$$\begin{bmatrix} 637.1 \\ 651.3 \\ 666.6 \end{bmatrix} = \begin{bmatrix} 0.49 & 0.39 & 0.12 \\ 0.06 & 0.93 & 0.004 \\ 0.00004 & 0.166 & 0.83 \end{bmatrix} \begin{bmatrix} P_{CPU,\,active} \\ P_{CPU,\,stall} \\ P_{CPU,\,idle} \end{bmatrix}$$

To obtain $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$, the above matrix is solved using LinearSolve function of Mathematica tool [12]. LinearSolve function solves the matrix for $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ (in this case 781.3 , 649.9 and 673.1 respectively). Similarly, each of the remaining equations obtained from the decoder program (From Table (1)) are solved with the equations from idle_test and stall_test programs (Equation (13) and (14) respectively). The resulting $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ values are shown in Table(2).

	FR (fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$P_{SoC,T}$ (mW)	$P_{CPU,active}$ (mW)	$P_{CPU,stall}$ (mW)	$P_{CPU,idle}$ (mW)
1	30	4cif	0.49	0.39	0.12	717.1	781.3	649.9	673.1
2	30	cif	0.16	0.26	0.59	689.3	817.3	647.6	673.6
3	30	qcif	0.07	0.22	0.71	679.2	837.0	646.3	673.8
4	25	4cif	0.26	0.35	0.38	708.8	852.7	645.3	674.0
5	25	cif	0.12	0.22	0.66	683.8	870.1	644.2	674.2
6	25	qcif	0.05	0.22	0.73	676.7	853.9	645.2	674.0
7	12.5	4cif	0.23	0.28	0.49	701.9	828.9	646.8	673.7
8	12.5	cif	0.09	0.20	0.71	684.3	855.3	645.1	674.1
9	12.5	qcif	0.03	0.20	0.77	674.2	873.2	644.0	674.3

Table 2: The calculated values for $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$

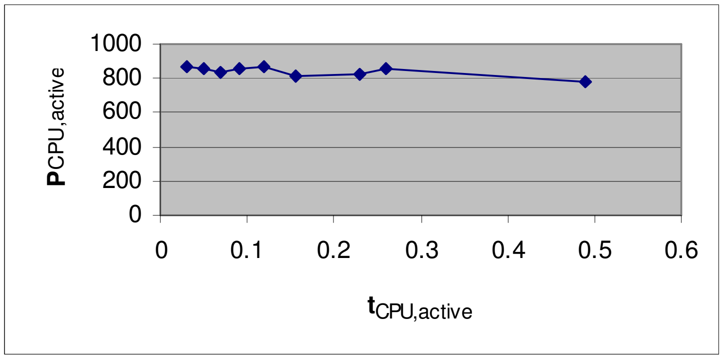
5.2.3.1 Analysis of results

Table (2) shows that the $t_{CPU,active}$ and $t_{CPU,stall}$ decrease with decrease in frame rate and frame size. This experimental result validates the assumption that the application dependent platform parameters depend on the application parameter settings. But the decrease in $t_{CPU,stall}$ with frame rate and frame size is not at the rate of decrease in $t_{CPU,active}$ with frame rate and frame size.

When the time spent by CPU in active state decreases, then the data cache misses as well as data cache stalls decrease. For example, in case of qcif resolution and 30fps in Table (2), CPU spends only 7% of the total time in active state ($t_{CPU,active}$). In this case it is expected that time spent in stall state ($t_{CPU,stall}$) is also relative to the $t_{CPU,active}$. But $t_{CPU,stall}$ is 22%. $t_{CPU,stall}$ is calculated as the sum of total instruction and data cache stalls. It was described in Section 5.2.1 that during idle task execution 16.6% are instruction cache stalls. Since in this example CPU spends 71% of the time in idle state, the instruction cache stalls during idle state are dominating in the $t_{CPU,stall}$. This explains why $t_{CPU,stall}$ is not scaling at the rate of $t_{CPU,active}$ with frame rate and frame size.

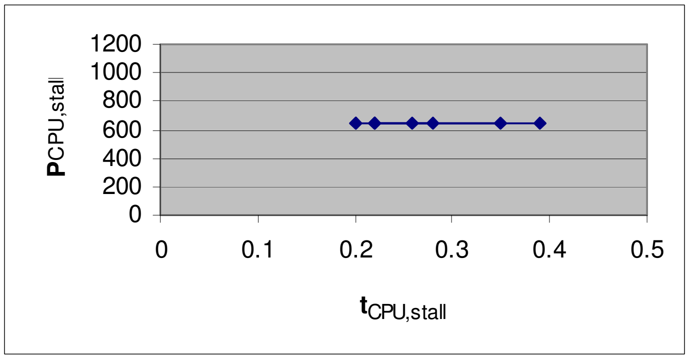
According to the assumption that the application independent platform parameters are independent of application parameters, the parameters $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ in Table (2) should be the same for any combination of frame rate and frame size. But, $P_{CPU,active}$ in Table (2) does not support this assumption. Even though $P_{CPU,stall}$ and $P_{CPU,idle}$ in Table (2) are not the same for each combination, the difference is very small.

Graph (1) shows the increase of $P_{CPU,active}$ with the decrease of time spent by CPU in active state ($t_{CPU,active}$). Graph (2) and (3) show that $P_{CPU,stall}$ and $P_{CPU,idle}$ are constant and does not vary with $t_{CPU,stall}$ and $t_{CPU,idle}$ respectively.

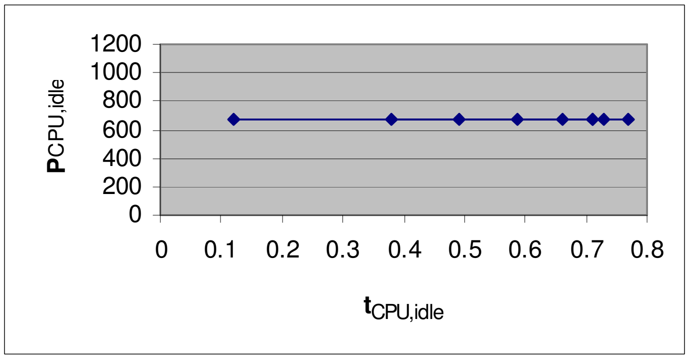


Graph 1: Graph representing $P_{CPU,active}$ vs. $t_{CPU,active}$

In the above graph, active power increases about 10%. We treat this increase of active power as an overhead in the active state. The reason for this overhead could be the assumption in Section 5.2 that the hardware blocks would be in active state only during CPU active periods. This overhead is more visible at less active periods of CPU. Section 5.4 and 5.4.1 explain about the overhead in detail.



Graph 2: Graph representing $P_{CPU,stall}$ vs. $t_{CPU,stall}$



Graph 3: Graph representing $P_{CPU,idle}$ vs. $t_{CPU,idle}$

Another observation from the results is that the power consumption during CPU idle state is more than the power consumption when CPU is in stall state (refer Table(2)). This is because of the fact that during idle state, CPU is not really idle but doing small amount of work during PSOS idle task (refer Section 5.2.1). $P_{CPU,idle}$ can be reduced if CPU goes in to power down mode during CPU idle state. Section 5.3 explains the CPU power down mode.

5.2.4 Linear regression

The second approach for obtaining $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ is to perform linear regression on the available data using the least square error method[13] in MATLAB. Linear regression method allows to find $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ values that fit all the linear equations considered. Linear regression is performed on the nine linear equaions obtained from the data in Table(1) for decoder program. Linear regression results in the follwing values.

$P_{CPU,active}$ = 609.9mW
$P_{CPU,stall}$ =889.2mW
$P_{CPU,idle}$ = 625.6mW

To measure the accuracy of the model, Root Mean Square Error is calculated [14]. The error obtained is 4.46mW. $P_{CPU,active}$(609.9mW) obtained is less than the stall power $P_{CPU,stall}$ (889.2mW). The reason is that the time spent by CPU in stall state is proportional to the time spent by CPU in active state. Therefore they both are correlated and are not independent enough to calculate the $P_{CPU,active}$ and $P_{CPU,stall}$ values separately. Therefore, the $t_{CPU,active}$ and $t_{CPU,stall}$ values in the equations are combined and are solved using linear regression method. The values thus obtained are as follows:

$P_{CPU,active+stall}$ = 732.9mW
$P_{CPU,idle}$ = 662.4 mW

The RMSE of the model with above coefficients is 5.7mW

If we perform linear regression on the nine linear equaions obtained from the data in Table(1) along with the idle_test and stall_test equations (Equation (13) and (14) respectively), it helps to calculate the $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ values more accurately, because these two tests characterize the power consumption of CPU in idle and stall states separately. Performing linear regression on the nine decoder equations along with idle_test and stall_test equations gives the following values:

$P_{CPU,active}$ = 792.7mW
$P_{CPU,stall}$ =664.4mW
$P_{CPU,idle}$ = 673.5mW

The RMSE of the model with above coefficients is 8mW

5.3 Experiments when CPU is in power down mode

In the latest version of NDK software (NDK 5.3), CPU power down mode feature is supported. The TM3260 CPU enters partial power down mode by performing a 'store' to a specific MMIO address (the POWERDOWN register). The TM3260 then finishes any pending transactions and goes into a partial power down. In partial power down mode, cycle counters, timers and interrupt logic in the TM3260 are still active. The TM3260CPU wakes up from partial power down when an interrupt occurs or there is an access to its MMIO space. Partial power down mode feature is used by the idle task in PSOS operating system [15]. It means that during the idle state of CPU, PSOS idle task makes CPU to enter into partial power down mode. In the previous sections NDK4.3 software was used, which does not have the CPU power down mode feature.

In the NDK4.3 version during the idle state of CPU, PSOS idle task is executed which is an infinite loop (refer Section 5.2.1).

The NDK5.3 software was installed and the experiments were done with decoder program with three different resolutions and frame rates. All the experiments were done at a CPU frequency of 100.5MHz. Table (3) shows the values obtained for $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$ for nine different experiments performed with decoder program.

	FR (fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$P_{SoC,T}$ (mW)
1	30	4cif	0.37	0.34	0.29	666.6
2	30	cif	0.13	0.15	0.72	575.7
3	30	qcif	0.07	0.09	0.83	550.4
4	25	4cif	0.30	0.30	0.40	643.9
5	25	cif	0.12	0.13	0.75	570.6
6	25	qcif	0.06	0.09	0.85	547.9
7	12.5	4cif	0.17	0.18	0.65	598.4
8	12.5	cif	0.07	0.09	0.84	558.1
9	12.5	qcif	0.04	0.06	0.90	540.4

Table 3: $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{CPU,idle}$ values obtained through various experiments in CPU power down mode

5.3.1 Power consumption in idle and stall states

Experiments done with idle_test program and stall_test program resulted in the following equations.

$$512.6 = P_{CPU,active} \times 0.00004 + P_{CPU,stall} \times 0.00034 + P_{CPU,idle} \times 0.999 \qquad (16)$$

$$651.4 = P_{CPU,active} \times 0.06 + P_{CPU,stall} \times 0.93 + P_{CPU,idle} \times 0.004 \qquad (17)$$

From Equation (16), it can be observed that the power consumption across SoC during idle_test ($P_{SoC,T}$ = 512.6) is reduced by 23.1% when compared to the power consumption ($P_{SoC,T}$ = 666.6) in without CPU power down mode. From Equation (16), it can be seen that CPU spends 99.9% of time in idle state. But, when CPU is not in power down mode, CPU spends only 83% (refer Equation (13)) of time in idle state and the remaining percentage of time (16.6%) in stall state. The 16.6% of stalls caused by the instruction cache misses during the idle task execution. Now, with CPU power down mode, idle task of PSOS makes CPU to enter into partial power down mode and there are no instruction cache misses. Therefore there are no stalls in this case.

Since CPU spends 99.9% of time in idle state during idle_test program, $P_{SoC,T}$ (512.6mW) from this test is taken as the power consumption when CPU is in idle state ($P_{CPU,idle}$).

5.3.2 Linear equation solutions

$P_{SoC,T}$ (512.6mW) from the idle_test is substituted in the $P_{CPU,idle}$ of the equations obtained from decoder and stall programs. Equation(18) is obtained by substituting $P_{CPU,idle}$ as 512.6mW, in the equation obtained by decoder program with an input stream of 4cif resolution and 30fps (refer Table(3)). Similarly, Equation (19) is obtained by substituting $P_{CPU,idle}$ as 512.6mW, in the equation of stall program (Equation(17)).

$$666.6 = P_{CPU,active} \times 0.37 + P_{CPU,stall} \times 0.34 + 512.6 \times 0.29 \quad (18)$$
$$651.4 = P_{CPU,active} \times 0.06 + P_{CPU,stall} \times 0.93 + 512.6 \times 0.004 \quad (19)$$

This results in the following set of equations:

$$517.9 = P_{CPU,active} \times 0.37 + P_{CPU,stall} \times 0.34 \quad (20)$$
$$649.3 = P_{CPU,active} \times 0.06 + P_{CPU,stall} \times 0.93 \quad (21)$$

The above equations can be written in the form of a matrix as shown below.

$$\begin{bmatrix} 517.9 \\ 649.3 \end{bmatrix} = \begin{bmatrix} 0.37 & 0.34 \\ 0.06 & 0.93 \end{bmatrix} \begin{bmatrix} P_{CPU,\,active} \\ P_{CPU,\,stall} \end{bmatrix}$$

To obtain $P_{CPU,active}$ and $P_{CPU,stall}$, the two linear equations (Equation (20) and (21)) are solved using LinearSolve function of Mathematica tool. LinearSolve function solves the matrix for $P_{CPU,active}$ and $P_{CPU,stall}$ (in this case 808 and 641.3 respectively). Similarly, each of the remaining decoder equations of Table (3) are solved with Eqaution (21). The resulting $P_{CPU,active}$ and $P_{CPU,stall}$ values are shown in the following table.

	FR(fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$P_{SoC,T}$-$P_{CPU,idle}$*$t_{CPU,idle}$ (mW)	$P_{CPU,active}$ (mW)	$P_{CPU,stall}$ (mW)
1	30	4cif	0.37	0.34	517.6	808.0	641.3
2	30	cif	0.13	0.15	207.1	843.3	638.8
3	30	qcif	0.07	0.09	122.9	893.8	636.3
4	25	4cif	0.30	0.30	438.3	815.6	641.3
5	25	cif	0.12	0.13	186.2	845.9	638.8
6	25	qcif	0.06	0.09	112.2	873.6	636.3
7	12.5	4cif	0.17	0.18	265.2	888.8	636.3
8	12.5	cif	0.07	0.09	129.5	992.3	628.7
9	12.5	qcif	0.04	0.06	80.8	1088.3	623.7

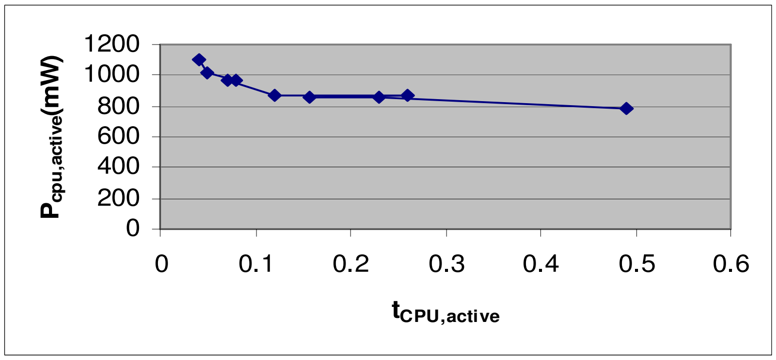
Table 4: The values calculated for $P_{CPU,active}$, $P_{CPU,stall}$ in CPU power down mode

5.3.2.1 Analysis of results

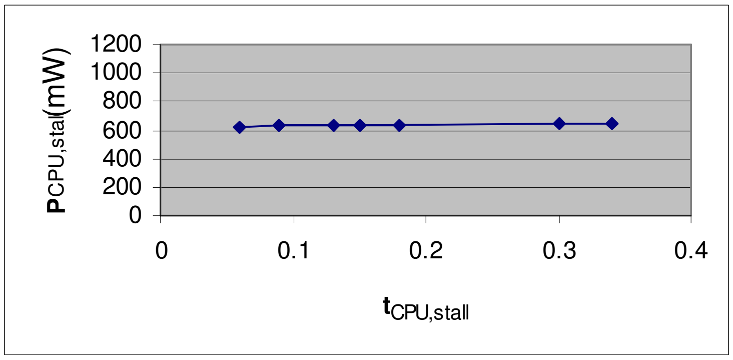
With CPU power down mode, idle power ($P_{CPU,idle}$ = 512.6) obtained is smaller than the stall power ($P_{CPU,stall}$ in Table (4)). This is not the case in Section 5.2.3 (without CPU power down), where the idle power is larger than the stall power in Table (2). The difference comes from the fact that during idle task execution CPU goes into partial power down mode with NDK5.3 version, whereas an infinite loop is executed in NDK4.3 version.

From the Table (4), it can be seen that the fraction of time CPU is in active and stall states ($t_{CPU,active}$ and $t_{CPU,stall}$) decreases with the decrease in frame rate and frame size, which supports the assumption that the application dependent platform parameters depend on the application parameters. It can also be observed that $t_{CPU,active}$ and $t_{CPU,stall}$, both scale almost at the same rate with frame rate and size. The reason for this is obvious because of the fact that there are no stalls in the idle state of CPU (refer Section 5.3.1). The stalls in the active state of CPU are directly proprtional to the time spent by CPU in active state.

Graph (4) shows that $P_{CPU,active}$ increases with the decrease in $t_{CPU,active}$. But $P_{CPU,stall}$ is independent of $t_{CPU,stall}$(refer Graph (4)). The systematic increase of active power in Graph (4) supports the observation in Section 5.2.3.1 that there is an overhead included in the CPU active state. As described earlier, Section 5.4 and 5.4.1 explain in detail about this overhead.



Graph 4: Graph representing $P_{CPU,active}$ vs. $t_{CPU,active}$



Graph 5: Graph representing $P_{CPU,stall}$ vs. $t_{CPU,stall}$

5.3.3 Linear regression

The second approach for obtaining $P_{CPU,active}$, $P_{CPU,stall}$ and $P_{CPU,idle}$ is to perform linear regression on the available data. Linear regression is performed on the nine linear equations obtained from the data in Table (3) for decoder program. Linear regression results in the follwing values.

$P_{CPU,active} = 799.5mW$
$P_{CPU,stall} = 648.1mW$
$P_{CPU,idle} = 523.6mW$

The Root Mean Squar Error obtained for the model is 2.9mW (refer Table(5)). The power consumption model for the SoC with above linear regression coefficients is given below:

$$P_{SoC,T} = 799.5 \times t_{CPU,active} + 648.1 \times t_{CPU,stall} + 523.6 \times t_{CPU,idle} \quad (22)$$

	FR(fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$P_{SoC,T}$ (actual) (mW)	$P_{SoC,T}$ (predicted) (mW)
1	30	4cif	0.37	0.34	0.29	666.6	668.0
2	30	cif	0.13	0.15	0.72	575.7	578.1
3	30	qcif	0.07	0.09	0.83	550.4	548.9
4	25	4cif	0.30	0.30	0.40	643.9	643.7
5	25	cif	0.12	0.13	0.75	570.6	572.9
6	25	qcif	0.06	0.09	0.85	547.9	551.3
7	12.5	4cif	0.17	0.18	0.65	598.4	592.9
8	12.5	cif	0.07	0.09	0.84	558.1	554.1
9	12.5	qcif	0.04	0.06	0.90	540.4	542.1
					Root Mean Square Error		2.9mW

Table 5: Actual and model predicted values for the $P_{SoC,T}$

The reason for obtaining larger values for $P_{CPU,stall}$ (648.1mW) and $P_{CPU,idle}$ (523.6mW) when compared to the $P_{CPU,stall}$ values in Table (4) and $P_{CPU,idle}$ (512.6mW) from idle_test program is explained as follows. From the Graphs (1) and (4), it was observed that there is some overhed in active state of CPU. But in linear regression approach, this overhead is distributed over the three states. Therefore, $P_{CPU,active}$ (799.5mW) is smaller and $P_{CPU,stall}$ and $P_{CPU,idle}$ are larger when compared to the results in the first approach.

5.4 Refined power consumption model

In the first approach, it was assumed that the hardware blocks are active only when CPU is in active state (refer Section 5.2). To validate this assumption, we need to monitor the behaviour of hardware blocks.

TimeDoctor tool is used to get the information about ISR (Interrupt service routine) CPU usage (refer Section 2.2.3). ISR informs CPU, whenever a hardware block is started and stopped (refer Figure (12)). Using the tmtdUserBlockCreate ( ), tmtdUserBlockEnter ( ) and tmtdUserBlockLeave ( ) API's of TimeDoctor tool [4], the execution cycles of hardware blocks are measured. From Figure (12), it can be seen that two hardware blocks: QVCP and MBS are active during decoder application. These hardware blocks are active periodically irrespective of the state of the CPU. Hence, the assumption that the hardware blocks are active only during active state of the CPU is not correct. The cost (power consumption) of active harware blocks is distributed over all the states of the CPU. But, in the first approach (refer Section 5.2), this overhead was included only in the active state of the CPU since the idle_test and stall_test do not include the power consumption due to active hardware blocks.

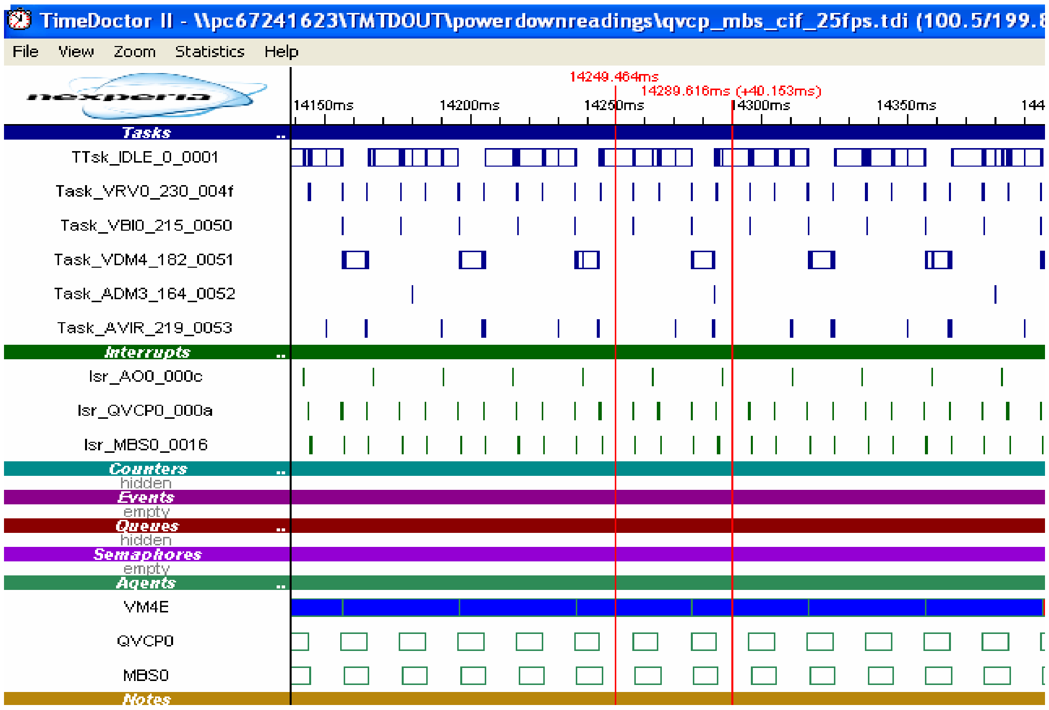


Figure 12: Screen shot of TimeDoctor viewer

In the second approach (Linear regression), the overhead due to hardware blocks is included in all the three states of the CPU. But, the actual values for $P_{CPU,active}$ $P_{CPU,stall}$ and $P_{CPU,idle}$ would be smaller than the values obtained in Section 5.3.3, if the power consumption model for SoC (Equation (12)) is included with the cost of the hardware blocks (QVCP and MBS) as well.

$P_{QVCP,active}$: Power consumed by QVCP block in active state
$P_{MBS,active}$: Power consumed by MBS block in active state
$P_{QVCP,idle}$: Power consumed by QVCP block in idle state
$P_{MBS,idle}$: Power consumed by QVCP block in idle state
$t_{QVCP,active}$: Fraction of time QVCP block is in active state
$t_{MBS,active}$: Fraction of time MBS block is in active state
$t_{QVCP,idle}$: Fraction of time QVCP block is in idle state
$t_{MBS,idle}$: Fraction of time MBS block is in idle state

$$P_{SoC,T} = P_{CPU,active} \times t_{CPU,active} + P_{CPU,stall} \times t_{CPU,stall} + P_{CPU,idle} \times t_{CPU,idle} + P_{QVCP,active} \times t_{QVCP,active} + P_{MBS,active} \times t_{MBS,active} + P_{QVCP,idle} \times t_{QVCP,idle} + P_{MBS,idle} \times t_{MBS,idle} \quad (23)$$

5.4.1 Experiments

Experiments were done with decoder program for three different resolutions and frame rates in CPU power down mode. Through TimeDoctor tool, the percentage of time spent by QVCP and MBS blocks in active state was calculated (refer Table (6)).

	FR (fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$t_{QVCP,active}$	$t_{MBS,active}$	$P_{SoC,T}$ (mW)
1	30	4cif	0.37	0.34	0.29	0.42	0.49	666.6
2	30	cif	0.13	0.15	0.72	0.43	0.42	575.7
3	30	qcif	0.07	0.09	0.83	0.43	0.42	550.4
4	25	4cif	0.30	0.30	0.40	0.43	0.50	643.9
5	25	cif	0.12	0.13	0.75	0.43	0.43	570.6
6	25	qcif	0.06	0.09	0.85	0.43	0.42	547.9
7	12.5	4cif	0.17	0.18	0.65	0.43	0.48	598.4
8	12.5	cif	0.07	0.09	0.84	0.43	0.42	558.1
9	12.5	qcif	0.04	0.06	0.90	0.43	0.42	540.4

Table 6: $P_{SoC,T}$, $t_{CPU, active}$, $t_{CPU, stall}$, $t_{CPU, idle}$, $t_{QVCP, active}$, $t_{MBS, active}$ values obtained through various experiments

From Table (6), it can be seen that the percentage of time spent by QVCP in active state ($t_{QVCP,active}$) is constant and does not vary with the frame rate and frame size. For MBS, percentage of time spent in active state ($t_{MBS,active}$) does not vary with frame rate also. But, $t_{MBS,active}$ for 4cif resolution is more when compared to cif and qcif resolutions. The QVCP and MBS blocks operate at the display frame rate i.e. at 50Hz. For example, for an input stream of frame rate 25fps, every 40ms a frame is executed, but QVCP and MBS blocks are executed twice in 40ms, which means at a frame rate of 50Hz (refer Figure (12)). Therefore, QVCP and MBS block executions are independent of the input frame rate. The $t_{QVCP,active}$ is independent of the input frame size, because the QVCP block processes all the pixels of the display resolution i.e. 4cif, irrespective of the input frame size. The MBS block does the image scaling by reading the video data from memory and writing the scaled pictures back to the memory. Since, MBS does the pixel based processing [15], $t_{MBS,active}$ depends on the input frame size.

The reason for obtaining larger values for $P_{CPU,active}$ shown in Graphs (1) and (4) when $t_{CPU,active}$ is small is explained as follows. The percentage of time spent by CPU in active state ($t_{CPU,active}$) decreases with the decrease in frame rate and frame size. But the time spent by QVCP and MBS blocks is constant with frame rate and frame size. Since the same amount of overhead is included in CPU active state irrespective of $t_{CPU,active}$, it is obvious that the overhead is more visible at small active percentages($t_{CPU,active}$).

5.4.1.1 Linear regression

The QVCP and MBS blocks have only two state active and idle. Therefore, $t_{QVCP,idle}$ and $t_{MBS,idle}$ values are calculated using the following equations:

$t_{QVCP,idle} = 1 - t_{QVCP,active}$
$t_{MBS,idle} = 1 - t_{MBS,active}$

The values obtained for $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$, $t_{CPU,idle}$, $t_{QVCP,active}$, $t_{MBS,active}$, $t_{QVCP,idle}$ and $t_{MBS,idle}$ for the nine different experiments are substituted in Equation(23). Solving the nine equations by linear regression gives the following results.

$P_{CPU,active}$=648.5mW
$P_{CPU,stall}$ = 318.0 mW

$P_{CPU,idle}$ = 307.3 mW
$P_{QVCP,active}$ =189.4 mW
$P_{MBS,active}$ = 52.3 mW
$P_{QVCP,idle}$ = 188.2 mW
$P_{MBS,idle}$ = 53.5 mW

The power consumtion model for SoC with the above regression coefficients is:

$$P_{SoC,T} = 648.5 \times t_{CPU,active} + 318.0 \times t_{CPU,stall} + 307.3 \times t_{CPU,idle} + 189.4 \times t_{QVCP,active} + 52.3 \times t_{MBS,active} + 188.2 \times t_{QVCP,idle} + 53.5 \times t_{MBS,idle} \quad (24)$$

The Root Mean Square Error of the above model is 2.13mW (refer Table 7). The above model is more accurate with 26% of reduction in RMSE when compared to the model in Section 5.3.3. But, at the same time the number of parameters of Equation (24) is doubled when compared to Equation (22) of Section 5.3.3.

	FR (fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$t_{QVCP,active}$	$t_{MBS,active}$	$t_{QVCP,idle}$	$t_{MBS,idle}$	$P_{SoC,T}$ (actual) (mW)	$P_{SoC,T}$ (predicted) (mW)
1	30	4cif	0.37	0.34	0.29	0.42	0.49	0.58	0.51	666.6	666.6
2	30	cif	0.13	0.15	0.72	0.43	0.42	0.57	0.58	575.7	574.6
3	30	qcif	0.07	0.09	0.83	0.43	0.42	0.57	0.58	550.4	550.4
4	25	4cif	0.30	0.30	0.40	0.43	0.50	0.57	0.50	643.9	645.0
5	25	cif	0.12	0.13	0.75	0.43	0.43	0.57	0.57	570.6	572.3
6	25	qcif	0.06	0.09	0.85	0.43	0.42	0.57	0.58	547.9	550.1
7	12.5	4cif	0.17	0.18	0.65	0.43	0.48	0.57	0.52	598.4	596.6
8	12.5	cif	0.07	0.09	0.84	0.43	0.42	0.57	0.58	558.1	553.5
9	12.5	qcif	0.04	0.06	0.90	0.43	0.42	0.57	0.58	540.4	542.9
								Root Mean Square Error			2.13mW

Table 7: Actual and model predicted values for $P_{SoC,T}$

5.4.2 Further simplification of the SoC power consumption model

The model presented in the last section is a good approximation for predicting power consumption, but it is not the simple model because of the number of parameters. Power consumed by QVCP and MBS blocks in idle state (parameters $P_{QVCP,idle}$ and $P_{MBS,idle}$) is the clock power of the blocks and can be measured in an experimental setup.

To measure the clock power, frequency of operation of these blocks during the execution of decoder program has to be known. The frequency of opeartion of these hardware blocks is obtained by reading the corresponding register values through URD tool, during the execution of decoder program. The opearting frequency of QVCP and MBS blocks are 27MHz and 108MHz respectively. The clock power of these blocks is calculated as follows:

At the reset position of the target, the clock frequency of the QVCP and MBS blocks is set as 27MHz and 108MHz respectively. The power consumption across SoC at this time is measured. Now the clock of the MBS block is disabled and then power consumption is measured. The difference between the two values gives the clock power of MBS block($P_{MBS,idle}$), which is 85.85mW. Now, the clock of the MBS block is enabled. The same procedure is followed to get the clock power of QVCP block, which is 22.7mW.

With the values of $P_{QVCP,idle}$ and $P_{MBS,idle}$, in Equation (23), the unknown parameters are reduced to five ($P_{CPU,active}$ $P_{CPU,stall}$ $P_{CPU,idle}$ $P_{QVCP,active}$ and $P_{MBS,active}$). $P_{QVCP,idle}$ and $P_{MBS,idle}$ are substituted in the Equations (23), the resulting equation is:

$$P_{SoC,T} - (22.7\times t_{QVCP,idle} + 85.85\times t_{MBS,idle}) = P_{CPU,active}\times t_{CPU,active} + P_{CPU,stall}\times t_{CPU,stall} + P_{CPU,idle}\times t_{CPU,idle} + P_{QVCP,active}\times t_{QVCP,active} + P_{MBS,active}\times t_{MBS,active} \quad (25)$$

Linear regression is performed on the nine equations obtained from the decoder program to get the parameters; $P_{CPU,active}$ $P_{CPU,stall}$ $P_{CPU,idle}$, $P_{QVCP,active}$ and $P_{MBS,active}$. Linear regression results in the following values for the parameters:

$P_{CPU,active}$= 645.9 mW
$P_{CPU,stall}$ = 316.1 mW
$P_{CPU,idle}$ = 305.1 mW
$P_{QVCP,active}$ = 157.5 mW
$P_{MBS,active}$ = 221.4 mW

The power consumtion model for the SoC with the above regression coefficients is:

$$P_{SoC,T} - (22.7\times t_{QVCP,idle} + 85.85\times t_{MBS,idle}) = 645.9\times t_{CPU,active} + 316.1\times t_{CPU,stall} + 305.1\times t_{CPU,idle} + 157.5\times t_{QVCP,active} + 221.4\times t_{MBS,active} \quad (26)$$

The RMSE of the above model is 2.13mW (refer Table (8)), which is equal to the RMSE of the previous model (Equation (24)).

	FR (fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$t_{QVCP,active}$	$t_{MBS,active}$	$P_{SoC,T}$ (actual) (mW)	$P_{SoC,T}$ (predicted) (mW)
1	30	4cif	0.37	0.34	0.29	0.42	0.49	609.6	609.6
2	30	cif	0.13	0.15	0.72	0.43	0.42	512.9	511.8
3	30	qcif	0.07	0.09	0.83	0.43	0.42	487.6	487.6
4	25	4cif	0.30	0.30	0.40	0.43	0.50	588.0	589.1
5	25	cif	0.12	0.13	0.75	0.43	0.43	508.7	510.4
6	25	qcif	0.06	0.09	0.85	0.43	0.42	485.1	487.3
7	12.5	4cif	0.17	0.18	0.65	0.43	0.48	540.8	539.0
8	12.5	cif	0.07	0.09	0.84	0.43	0.42	495.3	490.7
9	12.5	qcif	0.04	0.06	0.90	0.43	0.42	477.6	480.1
						Root Mean Square Error			2.13mW

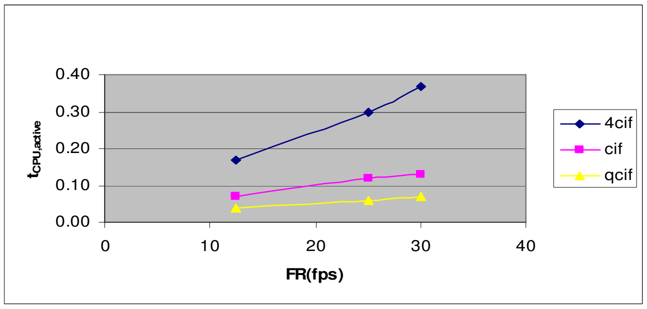
Table 8: Actual and model predicted values for $P_{SoC,T}$

5.5 *Models relating application dependent platform parameters to the application parameters*

It was observed from the results of Table(2) and (4) that the fraction of time spent by CPU in active and stall states ($t_{cpu,active}$ and $t_{cpu,stall}$) during the execution of decoder program decreases with the decrease in frame rate and frame size of the input stream. Time spent by QVCP block in active state $t_{QVCP, active}$ is independent of the input frame rate and frame size.Time spent by MBS block in active state ($t_{MBS, active}$) varies with the input frame size and is independent of the input frame rate (refer Table (6)). In this section the parameters $t_{cpu,active}$, $t_{cpu,stall}$ and $t_{MBS, active}$ are related to the application parameters. Through out this section the data from Table (8) is used to develop the models.

5.5.1 Models relating $t_{cpu,active}$ to the FR & FS

Graph (6) shows that $t_{cpu,active}$ increases linearly with frame rate for a constant frame size.

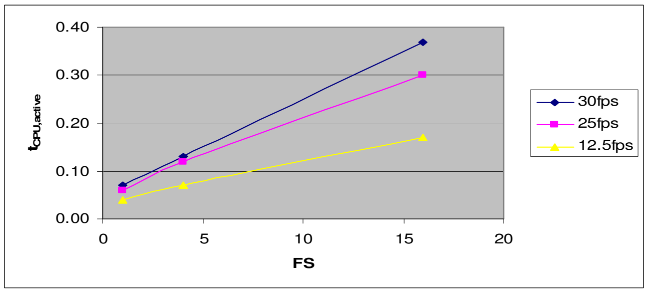


Graph 6: Graph representing $t_{cpu,active}$ vs. FR

Similarly, Graph (7) shows that, the $t_{cpu,active}$ increases linearly with frame size for a constant frame rate. In Graph (7), the values taken for the frame size are relative values not the absolute values.

Number of pixels for 4cif resolution is: 704×576
Number of pixels for cif resolution is: 352×288
Number of pixels for qcif resolution is: 176×144

Since number of pixels per frame increases four times from qcif to cif and similarly from cif to 4cif, the values taken for 4cif, cif and qcif in Graph (7) are 16, 4 and 1 respectively.



Graph 7: Graph representing $t_{cpu,active}$ vs. FS

5.5.1.1 Initial model (Model 1)

The $t_{cpu,active}$ varies linearly with the frame rate and frame size by keeping frame size and frame rate constant respectively. But, in Graph (6) the rate at which $t_{cpu,active}$ is increasing with frame rate is different for different frame size (number of pixels per frame). This is true for the Graph (7) as well. This suggest that the $t_{cpu,active}$, not only depends on frame rate and frame size individually but also on the combination of them. That means $t_{cpu,active}$ depends on the total number of pixels per second (FR*FS) as well. The following linear model is assumed to relate the $t_{cpu,active}$ to frame rate and frame size.

$$t_{CPU,active} = C1*FR*FS + C2*FR + C3*FS + C4 \qquad (27)$$

The $t_{cpu,active}$ measured for nine different combinations of frame rate and frame size (refer Table (8)) are substituted in the above equation:

$$\begin{aligned}
0.37 &= C1*480 + C2*30 + C3*16 + C4 \\
0.13 &= C1*120 + C2*30 + C3*4 + C4 \\
0.07 &= C1*30 + C2*30 + C3*1 + C4 \\
0.30 &= C1*400 + C2*25 + C3*16 + C4 \\
0.12 &= C1*100 + C2*25 + C3*4 + C4 \\
0.06 &= C1*25 + C2*25 + C3*1 + C4 \\
0.17 &= C1*200 + C2*12.5 + C3*16 + C4 \\
0.07 &= C1*50 + C2*12.5 + C3*4 + C4 \\
0.04 &= C1*12.5 + C2*12.5 + C3*1 + C4
\end{aligned}$$

Linear regression on the above nine equations result in the following regression coefficients:

C1 = 0.0006; C2 = 0.001; C3 = 0.0004; C4 = 0.021

With the above regression coefficients, the model for predicting the active percentage of CPU becomes:

$$t_{CPU,active} = 0.0006*FR*FS + 0.001*FR + 0.0004*FS + 0.021 \qquad (28)$$

	FR*FS	FR (fps)	FS	$t_{CPU,active}$ (actual)	$t_{CPU,active}$ (predicted)
1	480	30	16	0.37	0.36
2	120	30	4	0.13	0.13
3	30	30	1	0.07	0.07
4	400	25	16	0.30	0.31
5	100	25	4	0.12	0.11
6	25	25	1	0.06	0.06
7	200	12.5	16	0.17	0.17
8	50	12.5	4	0.07	0.07
9	12.5	12.5	1	0.04	0.04
			Root Mean Square Error		0.46%

Table 9: Actual and Model 1 predicted values for $t_{CPU,active}$

RMSE for the above model is calculated to be 0.46% (refer Table(9)). By normalizing the model presented in this section, it is also possible to compare the regression coefficients. The regression coefficients can be compared to see which term of the model has more influence on predicting the $t_{CPU,active}$.

5.5.1.2 Normalization of the Model 1

Following are the linear equations obtained by normalizing the model presented in the last section:

0.37 = C1*1 + C2*1 + C3*1 + C4
0.13 = C1*0.25 + C2*1 + C3*0.25 + C4
0.07 = C1*0.06 + C2*1 + C3*0.06 + C4
0.30 = C1* 0.83 + C2*0.83 + C3*1 + C4
0.12 = C1*0.208 + C2*0.83 + C3*0.25 + C4
0.06 = C1*0.05 + C2*0.83 + C3*0.06 +C4
0.17 = C1*0.417 + C2*0.417 + C3*1+ C4
0.07 = C1*0.1 + C2*0.417 + C3*0.25+ C4
0.04 = C1*0.03 + C2*0.417 + C3*0.06 +C4

The coefficients obtained by performing linear regression on the above equations are:

C1 = 0.305
C2 = 0.032
C3 = 0.008
C4 = 0.021

The coefficient C1 is larger than all other coefficients and this suggests that the term FR*FS has large influence in the model. This is also obvious from the fact that the term FR*FS (number of pixels per second) itself can capture the dependency of $t_{CPU,active}$ on FR and FS. The remaining coefficients are very small when compared to C1. Therefore, by eliminating all the terms except FR*FS term, we get a simplified model.

5.5.1.3 Simplified model (Model 2)

The following model includes only the term FR*FS

$$t_{CPU,active} = C1*FR*FS \quad (29)$$

The $t_{cpu,active}$ measured for nine different combinations of frame rate and frame size is substituted in the above equation. Linear regression on the equations of this model gives the coefficient C1, which is equal to 0.001. With this coefficient, the model becomes:

$$t_{CPU,active} = 0.001*FR*FS \quad (30)$$

Root Mean Square Error for this model is calculated to be 3.15% (refer Table (10)).

	FR*FS	$t_{CPU,active}$ (actual)	$t_{CPU,active}$ (predicted)
1	1	0.37	0.38
2	0.25	0.13	0.10
3	0.06	0.07	0.02
4	0.83	0.30	0.32
5	0.208	0.12	0.08
6	0.05	0.06	0.02
7	0.417	0.17	0.16
8	0.1	0.07	0.04
9	0.03	0.04	0.01
		Root Mean Square Error	3.15%

Table 10: Actual and Model 2 predicted values for $t_{CPU,active}$

5.5.1.4 Summary of models

The following table shows the summary of models considered so far and their corresponding Root Mean Square Errors.

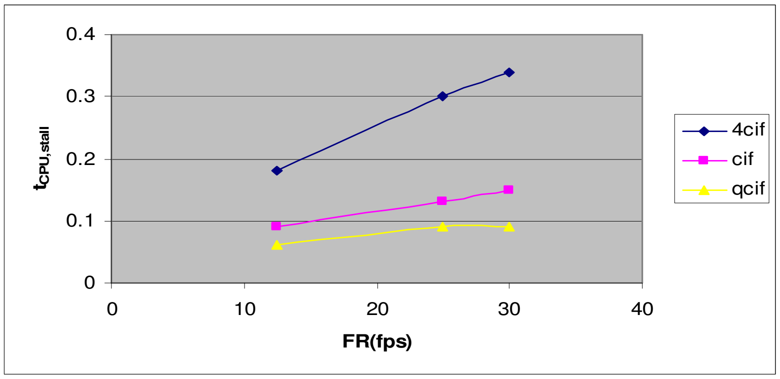
	Model	C1	C2	C3	C4	RMSE(%)
1	$t_{CPU,active}$ =C1*FR*FS+C2*FR+C3*FS+C4	0.0006	0.001	0.0004	0.021	0.46
1.a	Normalization of Model 1	0.305	0.032	0.008	0.021	0.45
2	$t_{CPU,active}$ =C1*FR*FS	0.0008				3.15
2.a	Normalization of Model 2	0.384				3.18

Table 11: Summary of the models and their RMSEs

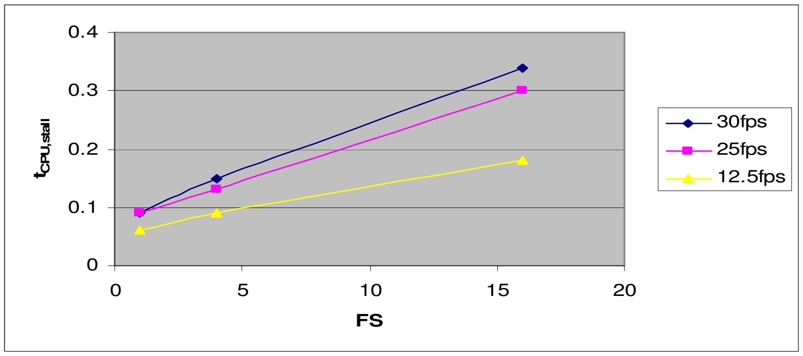
When we compare the two models in the above table, Model 2 has less number of parameters than Model 1 but the RMSE of Model 2 is much larger (7 times larger) than that of Model 1.

5.5.2 Model relating $t_{cpu,stall}$ to the FR and FS

Graphs (8) and (9) show that the $t_{cpu,stall}$ increases linearly with frame rate and frame size by keeping frame size and frame rate constant respectively. But here also, it can be seen that the rate of increase is not the same in all cases.



Graph 8: Graph representing $t_{CPU,stall}$ vs. FR



Graph 9: Graph representing $t_{CPU,stall}$ vs. FS

Different models considered in Section 5.5.1, to relate $t_{cpu,active}$ to the frame rate and frame size are considered in this section also to relate $t_{cpu,stall}$ to the frame rate and frame size.

Table (12) shows the regression coefficients obtained by solving linear equations and the RMSE of each model.

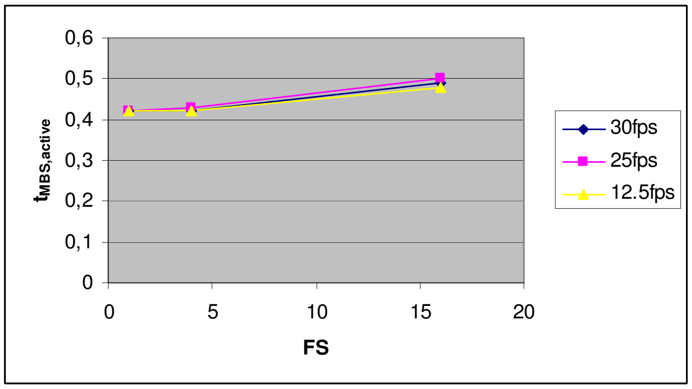
	Model	C1	C2	C3	C4	RMSE(%)
1	$t_{CPU,stall}$ =C1*FR*FS+C2*FR+C3*FS+C4	0.0005	0.001	0.001	0.04	0.29
1.a	Normalization of Model 1	0.239	0.044	0.024	0.04	0.33
2	$t_{CPU,stall}$ =C1*FR*FS	0.001				5.08
2.a	Normalization of Model 2	0.379				5.11

Table 12: Summary of the models and their RMSEs

The Model 2 of the above table is the simplest but the RMSE of this model is 15 times larger than the other. Model 1 is the most accurate model with small RMSE.

5.5.3 Model relating $t_{MBS,active}$ to the FS

From the results of Table (8), it can be seen that $t_{MBS, active}$ is independent of the input frame rate. But $t_{MBS,active}$ increases from the cif resolution to 4cif resolution. $t_{MBS, active}$ remains the same for cif and qcif resolutions (refer Graph (10)).



Graph 10: Graph representing $t_{MBS, active}$ vs. FS

The following model for relating the $t_{MBS, active}$ to the frame size is considered:

$t_{MBS,active} = C1 * FS + C2$ (31)

The values for the $t_{MBS,\ active}$ with an input stream of frame rate 30fps and resolution of 4cif , cif and qcif are substituted in the above model.

$0.49 = C1*16 + C2$
$0.42 = C1*4 + C2$
$0.42 = C1*1 + C2$

Normalizing the above equations gives the following equations:

$0.49 = C1*1 + C2$
$0.42 = C1*0.25 + C2$
$0.42 = C1*0.06 + C2$

Solving the above equations through linear regression gives the values for C1 and C2 as 0.08 and 0.41 respectively.

Therefore, the model for predicting the $t_{MBS,\ active}$ from frame size of the input stream is given in the following equation and the RMSE of the model is 0.6%

$t_{MBS,active} = 0.08 * FS + 0.41$ (32)

5.6 Compositional model for the white box approach

The method of nesting two or more functions to form a single new function is known as composition [16]. A compositional model for the white box approach is obtained by representing $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{MBS,active}$ of the SoC power model presented in Section 5.4.2, as a function of application parameters. With compositional model, we achieve a high level model that predicts the power consumption of the SoC from application parameters frame rate and frame size. Sections 5.5.1.4, 5.5.2 and 5.5.3 give the models that represent $t_{CPU,active}$, $t_{CPU,stall}$ and $t_{MBS,active}$ as a function of application parameters. The SoC power model presented in Section 5.4.2 is given below:

$P_{SoC,T} – (22.7\times t_{QVCP,idle} + 88.5\times t_{MBS,idle}) = 645.9 \times t_{CPU,active} + 316.1 \times t_{CPU,stall} + 305.1 \times t_{CPU,idle} + 157.5 \times t_{QVCP,active} + 221.4 \times t_{MBS,active}$

Where $t_{QVCP,active}$ is a constant value 0.43 from Table (8). From Table (11) and (12), the normalized models that relate $t_{CPU,active}$ and $t_{CPU,stall}$ to the FR and FS, with small RMSE are taken. Equation (32) represents $t_{MBS,active}$ as a function of FS. These models are given below:

$t_{CPU,active} = 0.3047*FR*FS + 0.032*FR + 0.0078*FS + 0.0206$
$t_{CPU,stall} = 0.239*FR*FS + 0.044*FR + 0.024*FS + 0.04$
$t_{MBS,active} = 0.08* FS + 0.41$
$t_{CPU,idle} = 1- t_{CPU,active} - t_{CPU,stall}$
$t_{QVCP,idle} = 1- 0.43 = 0.57$
$t_{MBS,idle} = 1- t_{MBS,active}$

Substituting the above equations in the SoC power model gives:

$$P_{SoC,T} = 119.8 \times FR*FS + 11.4 \times FR + 27.7\, FS + 536.3 \qquad (33)$$

The Root Mean Square Error of the above model is calculated to be 16mW, which is shown in the table below:

	FR*FS	FR	FS	$P_{SoC,T}$ (actual) (mW)	$P_{SoC,T}$ (predicted) (mW
1	1	1	1	666.6	695.2
2	0.25	1	0.25	575.7	584.6
3	0.06	1	0.06	550.4	556.6
4	0.83	0.83	1	643.9	672.9
5	0.21	0.83	0.25	570.6	577.6
6	0.05	0.83	0.06	547.9	553.4
7	0.42	0.42	1	598.4	618.7
8	0.1	0.42	0.25	558.1	560.0
9	0.03	0.42	0.06	540.4	546.3
		Root Mean Square Error			16mW

Table 13: Actual and compositional model predicted values for $P_{SoC,T}$

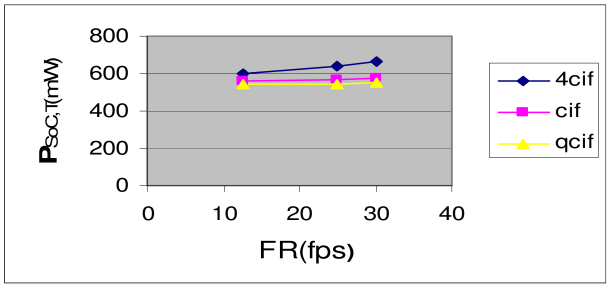
5.7 Black box approach to relate the average power consumption of SoC to FR and FS

The average power consumption measured across SoC for different settings of frame rate and frame size, with CPU power down mode are shown in the table below:

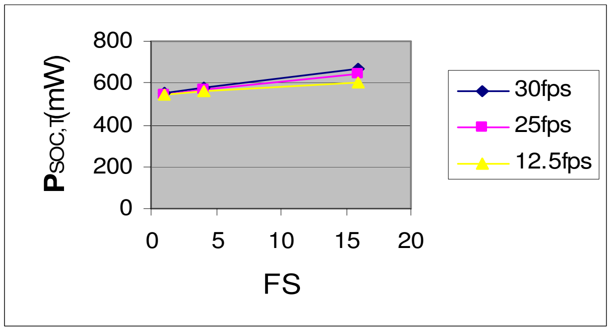
	FR(fps)	FS	$P_{SoC,T}$ (mW)
1	30	4cif	666.6
2	30	cif	575.7
3	30	qcif	550.4
4	25	4cif	643.9
5	25	cif	570.6
6	25	qcif	547.9
7	12.5	4cif	598.4
8	12.5	cif	558.1
9	12.5	qcif	540.4

Table 14: $P_{SoC,T}$ measured from different experiments

From the Graph (11), it can be seen that the average power consumption across SoC increases with the increase in frame rate by keeping frame size constant. But the rate of increase also depends on the frame size. This is also true for the Graph (12).



Graph 11: Graph representing $P_{SoC,T}$ vs. FR



Graph 12: Graph representing $P_{SoC,T}$ vs. FS

The following table shows various models considered for predicting the $P_{SoC,T}$ from FR and FS and their Root Mean Square Errors.

	Model	C1	C2	C3	C4	RMSE(mW)
1	$P_{SoC,T}$ =C1*FR*FS+C2*FR+C3*FS+C4	0.22	0.246	0.88	536.6	2.07
1.a	Normalization of Model 1	106.5	7.9	14.7	536.3	2.15
2	$P_{SoC,T}$ =C1*FR*FS+C4	0.26			542.8	2.73
2.a	Normalization of Model 2	124.3			542.9	2.80

Table 15: Summary of the models and their RMSEs

5.8 Comparison of the white box and black box models

This section compares the models obtained from the white box and black box approaches. The compositional model of the white box approach from Section 5.6 and the black box model from the Section 5.7 are given below:

$$P_{SoC,T} = 119.8*FR*FS + 11.4*FR + 27.7*FS + 536.3 \quad (34)$$

$$P_{SoC,T} = 106.5*FR*FS + 7.9*FR + 14.7*FS + 536.3 \quad (35)$$

The accuracy of the white box model is given by the RMSE of the model, which is equal to 16mW (refer Table (13)). The RMSE of the black box model is only 2.15mW (refer Table (15)). The RMSE of the white box approach is 7.4 times more than the RMSE of the black box approach. Therefore the black box models are more accurate than the white box models. The reason for large RMSE of the white box model is the method of composition of the models, in which the errors of individual models add up.

From both the models (white box and black box models), it can be observed that the term FR*FS (number of pixels per second) has large influence on the power consumption when compared to the terms: FR (number of frames per second) and FS (number of pixels per frame). This is obvious from the models of Table (11) and Table (12) that the term FR*FS has large influence on $t_{CPU,active}$ and $t_{CPU,stall}$ than the terms FR and FS. Number of pixels per frame (FS) of the input stream has more influence on the power consumption than the number of frames per second (FR). This is because of the fact that the execution periods of QVCP and MBS blocks are independent of FR but the execution period of MBS depends on FS (refer Section 5.4.1).

Another observation from the models is that there is large amount of constant offset power (536.3mW) consumed by the platform independent of the application parameters. This offset power is due to the clock power of the logic when the hardware components QVCP, MBS and various buses on the platform are in idle state.

6 Power models for memory

6.1 Introduction

This chapter presents the models to predict the average power consumption of the memory in two approaches: black box and white box. In the black box approach, a model that predicts the average power consumption of the memory directly from the application parameters, frame rate and frame size is developed through linear regression on experimental data. In white box approach, the power consumption of the memory is related to application parameters in two steps. The first step of this approach deals with experimentally validating the memory power consumption model presented in Section 4.4.2. The power consumption model for memory presented in Section 4.4.2 is given below:

$$P_{mem,T} = P_{mem,active} \times t_{mem,active} + P_{mem,idle} \times t_{mem,idle} + P_{mem,read} \times t_{mem,read} + P_{mem,write} \times t_{mem,write} \quad (36)$$

The application dependent platform parameters ($t_{mem,idle}$, $t_{mem,read}$ and $t_{mem,write}$) and $P_{mem,T}$ of the Equation (36) can be measured for a given frame rate and frame size of the input stream. Three different frame rates (30fps, 25fps and 12.5fps) and frame sizes (4cif, cif and qcif) are considered in the experiments. The application independent platform parameters ($P_{mem,active}$, $P_{mem,idle}$, $P_{mem,read}$ and $P_{mem,write}$) of the model are calculated by performing linear regression on the equations substituted with the experimentally measured values for parameters $t_{mem,active}$, $t_{mem,idle}$, $t_{mem,read}$ and $t_{mem,write}$ and $P_{mem,T.}$

In the second step of white box approach, a model relating the application dependent platform parameters to the application parameters is developed through linear regression on the experimental data. Finally, from the models of each step, a compositional model for the power consumption of the memory in terms of applcation parameters is developed. Using the compositional model, we can predict the average power consumption of the memory for any values of frame rate and frame size.

6.2 Black box approach experiments and results

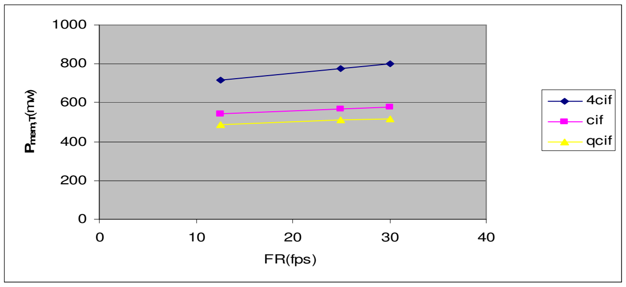
6.2.1 Without CPU power down

The average power consumption across the memory is measured during the execution of the decoder application for different values of frame rate (30fps, 25fps and 12.5fps) and frame size (4cif, cif and qcif) of the input stream. The maximum frequency of operation for the DDR memory is 199.8MHz. All the experiments are done at the maximum frequency.

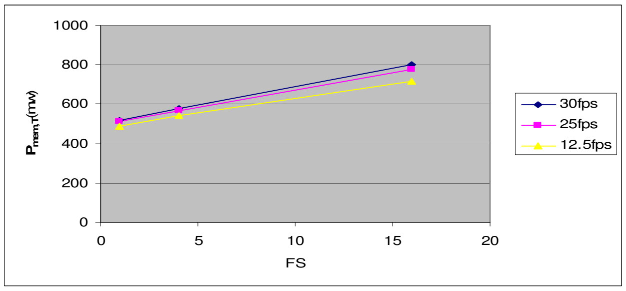
The following table shows the average power consumption measured for all the combinations of frame rates and frame sizes.

	FR(fps)	FS	$P_{mem,T}$(mW)
1	30	4cif	802
2	30	cif	575
3	30	qcif	517
4	25	4cif	775
5	25	cif	565
6	25	qcif	510
7	12.5	4cif	715
8	12.5	cif	540
9	12.5	qcif	490

Table 16: $P_{mem,T}$ values measured for different combinations of FR and FS



Graph 13: Graph representing $P_{mem,T}$ vs. FR



Graph 14: Graph representing $P_{mem,T}$ vs. FS

From the Graph (13), it can be seen that the average power consumption of the memory, $P_{mem,T}$ increases linearly with the frame rate by keeping frame size constant. But, the rate of increase in $P_{mem,T}$ depends on the frame size. The same is true for the Graph (14). Therefore the following model assumes three terms on which the average power consumption depends (FR*FS, FR and FS).

$$P_{mem,T} = C1*FR*FS + C2*FR + C3*FS + C4 \qquad (37)$$

$P_{mem,T}$ measured for the different combinations of FR and FS from Table (16) is substituted in the above equation. Linear regression on the equations give the following coefficients:

C1 = 0.23; C2 = 1.22; C3 = 11.95; C4 = 462.31

With the above regression coefficients the model for predicting the average power consumption becomes:

$$P_{mem,T} = 0.23*FR*FS + 1.22*FR + 11.95*FS + 462.31 \qquad (38)$$

The Root Mean Square Error obtained for the above model is 1.54mW.

Linear regression on the normalized equations of the above model (Equation (37)) gives the following regression coefficients:

C1 = 110.5; C2 = 37.35; C3 = 191.4; C4 = 462.3

Equation(37) can be simplified by removing the term FR, since it has less influence on predicting the average power consumption when compared to the other terms. The simplified model along with the initial model and their RMSEs are presented in Table (17).

	Model	C1	C2	C3	C4	RMSE(mW)
1	$P_{mem,T}$ =C1*FR*FS+C2*FR+C3*FS+C4	0.23	1.22	11.95	462.3	1.54
1.a	Normalization of Model 1	110.5	37.3	191.4	462.3	1.55
2	$P_{mem,T}$ =C1*FR*FS+C3*FS+C4	0.32		9.83	489.9	6.31
2.a	Normalization of Model 2	156.2		157.2	490.2	6.43

Table 17: Summary of the models and their RMSEs

When we compare the models in the above table, Model 1 has more accuracy with small error. Model 2 has less number of parameters, but the error is 4 times larger than the error of Model 1.

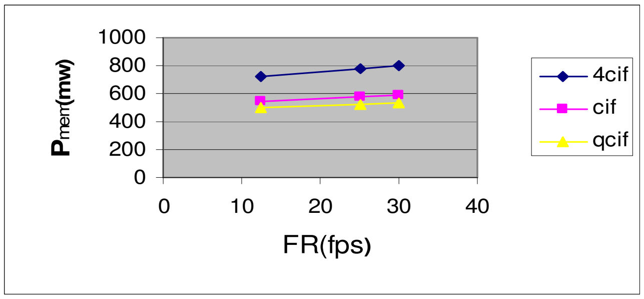
6.2.2 With CPU power down

The following table shows the average power consumption measured across the memory, $P_{mem,T}$ for different combinations of frame rate and frame size with CPU power down mode.

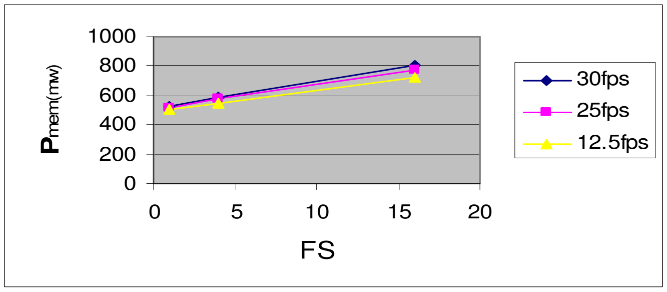
	FR(fps)	FS	$P_{mem,T}$ (mW)
1	30	4cif	805
2	30	cif	585
3	30	qcif	528
4	25	4cif	777
5	25	cif	578
6	25	qcif	517
7	12.5	4cif	724
8	12.5	cif	548
9	12.5	qcif	503

Table 18: $P_{mem,T}$ values measured for different combinations of FR and FS

The following graphs show that the $P_{mem,T}$ increases linearly with the FR and FS



Graph 15: Graph representing $P_{mem,T}$ vs. FR



Graph 16: Graph representing $P_{mem,T}$ vs. FS

The following table shows various models considered for predicting the $P_{mem,T}$ from FR and FS and their Root Mean Square Errors.

	Model	C1	C2	C3	C4	RMSE(mW)
1	$P_{mem,T}$ =C1*FR*FS+C2*FR+C3*FS+C4	0.21	1.24	12.06	473.4	2.53
1.a	Normalization of Model 1	99.73	37.6	192.9	473.4	2.34
2	$P_{mem,T}$ =C1*FR*FS+C3*FS+C4	0.3		9.92	501.2	6.7
2.a	Normalization of Model 2	145.7		158.5	501.6	6.7

Table 19: Summary of the models and their RMSEs

From the above table, it can be seen that Model 1 has small RMSE and thus has more accuracy. Model 2 is simpler than Model 1 with less number of parameters but the RMSE of Model 2 is 2.9 times larger than the RMSE of Model 1.

6.2.2.1 Comparison of black box models of SoC and memory (with CPU power down)

This section compares the power models of SoC and memory obtained with black box approach. Table (15) of Chapter 5 gives the power model of SoC with black box approach. We chose the normalized model with small RMSE (2.15mW) from Table (15) and the model is given below:

$$P_{SoC,T} = 106.5*FR*FS + 7.9*FR + 14.7*FS + 536.3$$

From Table (19), we chose the normalized power model of memory with small RMSE (2.34mW). The model is given below:

$$P_{mem,T} = 99.7*FR*FS + 37.6*FR + 192.9*FS + 473.4$$

Both the models suggest that the term FR has less influence on the power consumption of the SOC and memory. The term FS (number of pixels per frame) has large influence on the power consumption of memory, whereas the term FR*FS has large influence on the power consumption of SoC.

On SoC side, time spent by CPU in active state ($t_{CPU,active}$) is more influenced by the term FR*FS than the other terms (refer Section 5.5.1.4). The hardware blocks QVCP and MBS executions are not influenced by FR. The execution periods of MBS block depends only on FS (refer Section 5.4.1). The large influence of FR*FS term on the power consumption of the SoC indicates that the CPU has more influence on the power consumption of SoC when compared to the hardware blocks. The large influence of FS term on the power consumption of the memory indicates that the hardware blocks (specially MBS block) have more influence on the power consumption of memory than the CPU.

6.3 White box approach experiments and results

6.3.1 Measurement of application dependent platform parameters through experiments

The data path width of the DDR memory is 32-bit. The DDR controller provides an interface between CPU, DMA devices and the DDR memory. To allow for the performance measurements, the DDR controller includes a set of registers that measure the data traffic [15]. To measure the read and write traffic from CPU as well as from DMA devices, incrementing 32-bit counters are used. The controller also includes a counter to count the idle cycles. The TimeDoctor tool is used to collect the values of these counters.

During the execution of the decoder application, the values from the abovementioned counters are read through the TimeDoctor tool. Table (20) shows the $P_{mem,T}$ and the counter values measured for different combinations of frame rate and frame size without CPU power down mode. Similarly, Table (21) shows the $P_{mem,T}$ and the counter values with CPU power down mode. The experiments were done at a memory frequency of 199.8 MHz and at a CPU frequency of 100.5 MHz (refer Section 5.2).

	FR (fps)	FS	CPU_{read} (Mcy)	CPU_{write} (Mcy)	DMA_{read} (Mcy)	DMA_{write} (Mcy)	DDR_{idle} (Mcy)
1	30	4cif	7.38	4.46	6.30	4.98	167.8
2	30	cif	3.50	1.46	1.56	1.20	189.8
3	30	qcif	2.58	0.73	0.45	0.32	193.8
4	25	4cif	6.49	3.75	6.33	4.99	171.8
5	25	cif	3.07	1.25	1.62	1.25	189.8
6	25	qcif	2.32	0.65	0.45	0.32	193.8
7	12.5	4cif	3.96	2.09	6.31	4.98	177.8
8	12.5	cif	2.09	0.76	1.63	1.25	191.8
9	12.5	qcif	1.61	0.43	0.45	0.32	195.8

Table 20: Read and write cycles from CPU and DMA devices measured through TimeDoctor tool in different experiments without CPU power down mode

	FR (fps)	FS	CPU_{read} (Mcy)	CPU_{write} (Mcy)	DMA_{read} (Mcy)	DMA_{write} (Mcy)	DDR_{idle} (Mcy)
1	30	4cif	6.26	4.94	6.26	4.94	167.8
2	30	cif	1.61	1.23	1.61	1.23	187.8
3	30	qcif	0.45	0.37	0.45	0.37	193.8
4	25	4cif	6.39	5.03	6.35	5.00	171.8
5	25	cif	1.63	1.25	1.63	1.25	189.8
6	25	qcif	0.45	0.32	0.45	0.32	193.8
7	12.5	4cif	6.31	4.97	6.31	4.97	177.8
8	12.5	cif	1.62	1.24	1.62	1.24	191.8
9	12.5	qcif	0.45	0.31	0.45	0.31	195.8

Table 21: Read and write cycles from CPU and DMA devices measured through TimeDoctor tool in different experiments with CPU power down mode

In Table (20), the read and write accesses from CPU to memory increase with the increase in frame rate and frame size. This experimental result validates the assumption that the application dependent platform parameters depend on the application parameters (refer section 6.1). But, the read and write accesses from DMA traffic i.e. read and write accesses from QVCP and MBS blocks increase with the increase in frame size but are independent of changes in frame rate (refer Section 5.4.1).

It is expected that the read and write accesses from CPU and DMA devices with CPU power down mode also have the same relation with the application parameters as for without CPU power down mode. But from the Table (21), it can be seen that read and write accesses from CPU do not vary with the frame rate. The statistics given by TimeDoctor tool for frame rates 30fps, 25fps and 12.5fps at a frame size of 4cif are given in Table (22). From the statistics, it can be seen that the reads and writes from CPU during the decoding task (TASK_VDM4_182_0051 in Figure (10)) decrease with the decrease in frame rate. But, there are reads and writes from the CPU during the execution of idle task (IDLE in Figure (10)) and the number of read and write cycles are increasing when the frame rate is decreasing. This increase in read and write cycles with the decrease in frame rate during idle task compensates the normal effect of decrease in read and write accesses with frame rate during decoding task. Because of this, in Table (21) we see no dependency of read and write accesses on frame rate.

			idle task		decoding task		idle+decoding tasks	
	FR(fps)	FS	CPU_{read} (Mcy)	CPU_{write} (Mcy)	CPU_{read} (Mcy)	CPU_{write} (Mcy)	CPU_{read} (Mcy)	CPU_{write} (Mcy)
1	30	4cif	1.2	0.8	4.7	3.9	5.8	4.7
2	25	4cif	1.8	1.2	4.1	3.5	5.9	4.7
3	12.5	4cif	3.8	2.9	2.2	1.8	5.9	4.8

Table 22: CPU read/write cycles from/to the memory during idle and decoding tasks

We measured the number of read and write misses ($miss_{read}$ and $miss_{write}$ in Table (23)) from the instruction and data cache of the CPU (using TM3260 CPU counters) during idle task for the frame rate and frame size given in Table (22). The measured values are given Table (23). There are very few read and write misses from the CPU during idle task.

	FR(fps)	FS	$miss_{read}$ (M)	$miss_{write}$ (M)	CPU_{read} (Mcy)	CPU_{write} (Mcy)
1	30	4cif	0.03	0.001	1.2	0.8
2	25	4cif	0.08	0.001	1.8	1.2
3	12.5	4cif	0.08	0.001	3.8	2.9

Table 23: Cache read/write misses and CPU read/write cycles from/to the memory during idle task

Analysis of the relation between the number of cache read/write misses and CPU read/write cycles from/to the memory are beyond the scope of this work. Hence, in the rest of this chapter, the white box models are developed considering the measurement data from without CPU power down experiments.

6.3.2 Calculation of application independent platform parameters through linear regression

For the decoder application a burst length of 8 is used. Burst length can be set through the registers of the DDR controller. Since the data path width of DDR memory is 32-bit, the burst size is 32B and therefore 8 words. The counters for measuring read and write data are incremented by 32, which means the values in these counters are the number of read and write words. Since, DDR memory can output 32-bit data per cycle, the read and write words from the counters can also be represented as read and write cycles. Therefore, the TimeDoctor tool gives the values from the counters as read and write cycles. Number of bursts in a given stream can be calculated by dividing the sum of read and write words from the counters with s_{burst} i.e. 8 words. But the number of cycles taken by the memory for activate-precharge (*burst*) activity is not known.

As described in Section 4.4.2, from the parameters that can be measured experimentally, the following model is considered:

$$P_{mem,T} = P_{mem,active} \times t_{mem,active} + P_{mem,idle} \times t_{mem,idle} + P_{mem,read} \times t_{mem,read} + P_{mem,write} \times t_{mem,write} \qquad (39)$$

$$t_{mem,active} = 1 - t_{mem,idle} \qquad (40)$$

The $t_{mem,active}$ is calculated as the difference of total time and idle time. $t_{mem,idle}$, $t_{mem,read}$ and $t_{mem,write}$ of Equation (39) are calculated from the idle, read and write cycles of Table(20). $t_{mem,active}$ is calculated by using Equation (40). The values calculated for $t_{mem,idle}$, $t_{mem,read}$,$t_{mem,write}$ and $t_{mem,active}$ for different combinations of frame rate and frame size are given in the Table(24). Table also shows the $P_{mem,T}$ measured for each combination.

	FR (fps)	FS	$t_{mem,idle}$	$t_{mem,active}$	$t_{mem,read}$	$t_{mem,write}$	$P_{mem,T}$ (mW)
1	30	4cif	0.84	0.16	0.069	0.047	802
2	30	cif	0.95	0.05	0.025	0.013	575
3	30	qcif	0.97	0.03	0.015	0.005	517
4	25	4cif	0.86	0.14	0.060	0.044	775
5	25	cif	0.95	0.05	0.020	0.013	565
6	25	qcif	0.97	0.03	0.014	0.005	510
7	12.5	4cif	0.89	0.11	0.051	0.035	715
8	12.5	cif	0.96	0.04	0.019	0.010	540
9	12.5	qcif	0.98	0.02	0.010	0.004	490

Table 24: $t_{mem,idle}$, $t_{mem,active,}$ $t_{mem,read}$ and $t_{mem,write}$ values calculated for different combinations of FR and FS

The values for $P_{mem,T,}$ $t_{mem,idle}$, $t_{mem,read}$, $t_{mem,write}$ and $t_{mem,active}$ from the nine different experiments are substituted in the Equation (39). Performing linear regression on the nine equations gives the following values for $P_{mem,active}$, $P_{mem,,idle}$, $P_{mem,read}$ and $P_{mem,write}$.

$P_{mem,active}$ = 829.9 mW
$P_{mem,,idle}$ = 462.5mW
$P_{mem,read}$ = 1068.9mW
$P_{mem,write}$ = 4466.1mW

The RMSE obtained for the model (Equation (39)) with the above coefficients is 4.07mW (refer Table (25)).

	FR (fps)	FS	$t_{mem,idle}$	$t_{mem,active}$	$t_{mem,read}$	$t_{mem,write}$	$P_{mem,T(measured)}$ (mW)	$P_{mem,T(predicted)}$ (mW)
1	30	4cif	0.84	0.16	0.069	0.047	802	806
2	30	cif	0.95	0.05	0.025	0.013	575	571
3	30	qcif	0.97	0.03	0.015	0.005	517	517
4	25	4cif	0.86	0.14	0.060	0.044	775	775
5	25	cif	0.95	0.05	0.020	0.013	565	560
6	25	qcif	0.97	0.03	0.014	0.005	510	506
7	12.5	4cif	0.89	0.11	0.051	0.035	715	713
8	12.5	cif	0.96	0.04	0.019	0.010	540	543
9	12.5	qcif	0.98	0.02	0.010	0.004	490	498
					Root Mean Square Error			4.07mW

Table 25: Measured and model predicted values for $P_{mem,T}$ and the corresponding RMSE

6.3.3 Simplified model

The values for $t_{mem,read}$ and $t_{mem,write}$ (refer Table (24)) are small and are closely related to each other. Since $t_{mem,read}$ and $t_{mem,write}$ are not independent enough it is difficult to distinguish the read and write power consumption ($P_{mem,read}$ and $P_{mem,write}$) separately through linear regression. Therefore the Equation (39) is further simplified by combining the $t_{mem,read}$ and $t_{mem,write}$ as shown below.

$$P_{mem,T} = P_{mem,active} \times t_{mem,active} + P_{mem,idle} \times t_{mem,idle} + P_{mem,read\&write} \times t_{mem,read\&write} \quad (41)$$

The values for $t_{mem,active}$,$t_{mem,idle}$ and $t_{mem,read\&write}$ from Table (24) are substituted in the above model. Performing linear regression on the obtained linear equations gives the following values for $P_{mem,active}$, $P_{mem,,idle}$ and $P_{mem,read\&write}$.

$P_{mem,active}$ = 1600.4mW
$P_{mem,,idle}$ = 450.5mW
$P_{mem,read\&write}$ = 1526.8mW

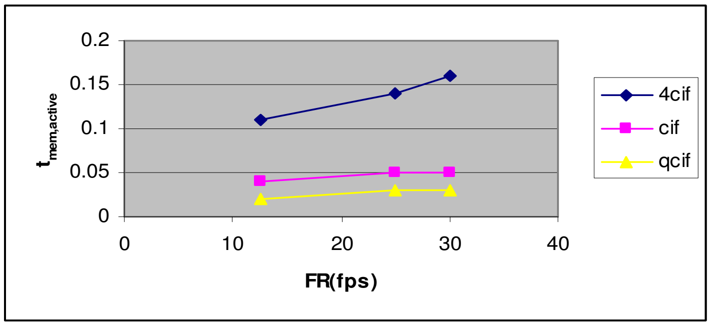
The RMSE of the above memory model with the regression coefficients is calculated to be 5.91mW which is 1.5 times larger than the RMSE (4.07mW) of the previous model (Equation(39)). But, the simplified model has less number of parameters when compared to the previous model.

6.4 Models relating application dependent platform parameters to application parameters

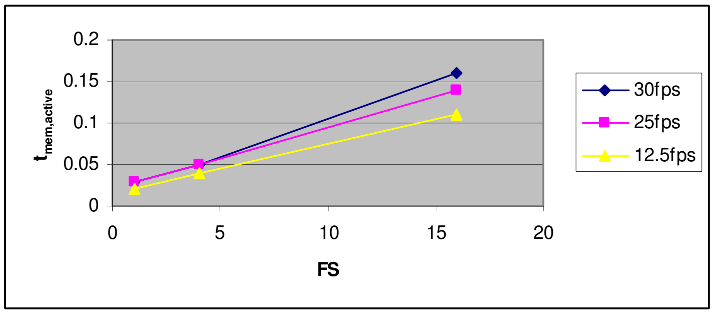
The second step of the white box approach is to relate the application dependent platform parameters ($t_{mem,active}$, $t_{mem,read}$ and $t_{mem,write}$) to the application parameters frame rate and frame size. In this section, models relating the application dependent platform parameters to the application parameters are developed.

6.4.1 Models relating $t_{mem,active}$ to the FR and FS

The following graphs show that the $t_{mem,active}$ increases linearly with FR and FS by keeping the other parameter constant. The values for $t_{mem,active}$ are taken from the Table (24).



Graph 17: Graph representing $t_{mem,active}$ vs. FR



Graph 18: Graph representing $t_{mem,active}$ vs. FS

From the above graphs, it can be seen that $t_{mem,active}$ depends on both FR and FS. For example, in Graph (17), the rate of increase of $t_{mem,active}$ with FR is more for 4cif resolution than for cif and qcif resolutions. Similarly, the rate of increase of $t_{mem,active}$ with FS (refer Graph (18)) is more for 30fps than for 25fps and 12.5fps. Therefore, to predict $t_{mem,active}$ from FR and FS, the following linear model is considered.

$$t_{mem,active} = C1*FR*FS + C2*FR + C3*FS + C4 \qquad (42)$$

The values for $t_{mem,active}$ from Table(24) for different values of FR and FS are substituted in the above equation. The linear equations thus obtained are solved through linear regression. The following coefficients are obtained from linear regression.

C1 = 0.0002; C2 = 0.0003; C3 =0.004; C4 = 0.013

In order to make a comparison between the coefficients in terms of their influence on predicting the $t_{mem,active}$, the equations obtained from the above model are normalized. Performing linear regression on the normalized equations gives the following coefficients.

C1 = 0.07; C2 = 0.008; C3 =0.063; C4 = 0.0126

The RMSE obtained for the model with above regression coefficients is 0.25%. From the above coefficients, it can be seen that the terms FR*FS (C1) and FS (C3) have large impact on $t_{mem,active}$ than the term FR (C2) and the constant (C4). Therefore Equation (42) can be simplified by removing the terms that have less influence on $t_{mem,active.}$ In the following model, only the terms FR*FS and FS are considered.

$$t_{mem,active} = C1*FR*FS + C3*FS \qquad (43)$$

Linear regression on the equations obtained by substituting $t_{mem,active}$, FR and FS from Table(24) in the above model gives the following coefficients.

C1 = 0.0002; C3 =0.005

Performing linear regression on the normalized equations of the simplified model gives the following coefficients.

C1 = 0.08; C2 = 0.08

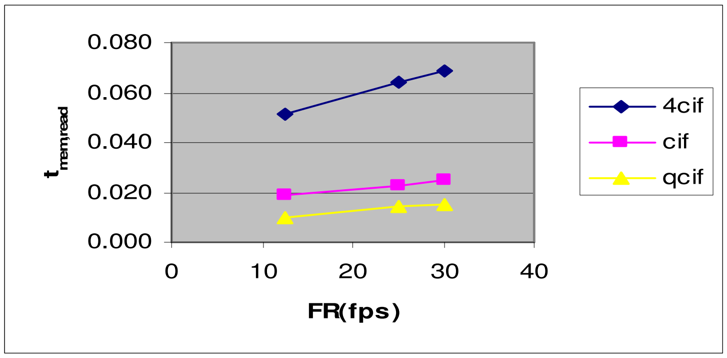
The RMSE obtained for the simplified model with the above regression coefficients is 1.3% which is 5.2 times larger than the RMSE (0.25%) of the previous model (Equation (42)). The following table shows the summary of the models considered and their Root Mean Square Errors.

	Model	C1	C2	C3	C4	RMSE(%)
1	$t_{mem,active}$ = C1*FR*FS+C2*FR+C3*FS+C4	0.0002	0.0003	0.004	0.013	0.25
1.a	Normalization of Model 1	0.07	0.008	0.063	0.013	0.25
2	$t_{mem,active}$ = C1*FR*FS+C3*FS	0.0002		0.005		1.3
2.a	Normalization of Model 2	0.08		0.08		1.3

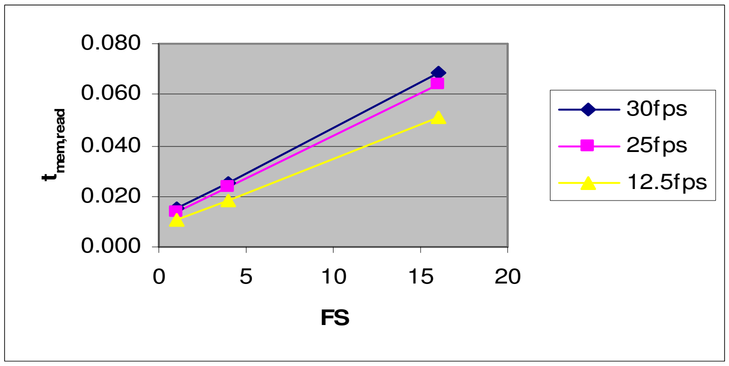
Table 26: Summary of models and their RMSEs

6.4.2 Models relating $t_{mem,read}$ to the FR and FS

The values for $t_{mem,read}$ for different combinations of FR and FS are taken from the Table (24). The following graphs show that $t_{mem,read}$ increases linearly with FR and FS by keeping the other parameter constant.



Graph 19: Graph representing $t_{mem,read}$ vs. FR



Graph 20: Graph representing $t_{mem,read}$ vs. FS

The following table shows the models considered for predicting the $t_{mem,read}$ from FR and FS and their Root Mean Square Errors.

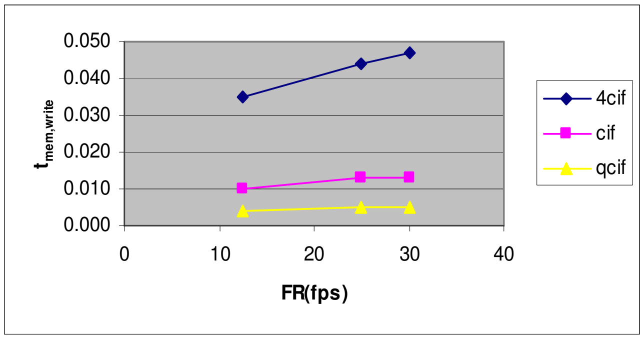
	Model	C1	C2	C3	C4	RMSE(%)
1	$t_{mem,read}$ = C1*FR*FS+C2*FR+C3*FS+C4	0.0001	0.0002	0.002	0.005	0.04
1.a	Normalization of Model 1	0.02	0.006	0.03	0.005	0.04
2	$t_{mem,read}$ = C1*FR*FS+C3*FS	0.0001		0.0025		0.7
2.a	Normalization of Model 2	0.03		0.04		0.7

Table 27: Summary of models and their RMSEs

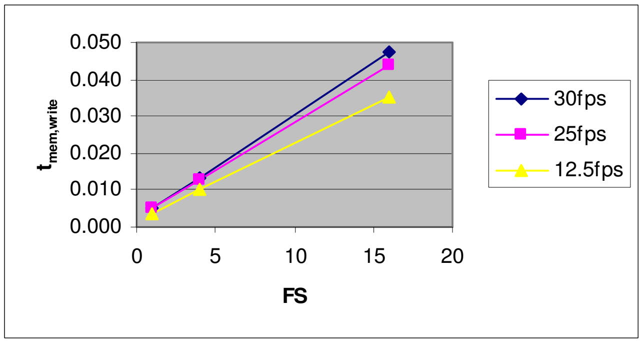
When we compare the models in the above table, the RMSE of the Model 2 is 17.5 times larger than the RMSE of the Model 1 with only reduction in two parameters.

6.4.3 Models relating $t_{mem,write}$ to the FR and FS

From the graphs below, it can be seen that the $t_{mem,write}$ without CPU power down increases linearly with FR and FS.



Graph 21: Graph representing $t_{mem,write}$ vs. FR



Graph 22: Graph representing $t_{mem,write}$ vs. FS

The following table shows the models considered for predicting the $t_{mem,write}$ from FR and FS and their Root Mean Square Errors.

	Model(Normalized)	C1	C2	C3	C4	RMSE(%)
1	$t_{mem,write}$ = C1*FR*FS+C2*FR+C3*FS+C4	0.02	0.0006	0.02	0.002	0.03
2	$t_{mem,write}$ = C1*FR*FS+C3*FS	0.02		0.03		0.15

Table 28: Summary of models and their RMSEs

6.5 Compositional model for the white box approach

A compositional model is obtained by representing $t_{mem,active}$, $t_{mem,read}$ and $t_{mem,write}$ of the memory power model presented in Section 6.3.2, as a function of application parameters. With compositional model, we achieve a high level model that predicts the power consumption of the memory from application parameters, frame rate and frame size. Sections 6.4.1, 6.4.2 and 6.4.3 give the models that represent $t_{mem,active}$, $t_{mem,read}$ and $t_{mem,write}$ as a function of application parameters. The memory power model presented in Section 6.3.2 is given below:

$$P_{mem,T} = P_{mem,active} \times t_{mem,active} + P_{mem,idle} \times t_{mem,idle} + P_{mem,read} \times t_{mem,read} + P_{mem,write} \times t_{mem,write} \quad (44)$$

From Table (26), (27) and (28), the normalized models which relate $t_{mem,active}$, $t_{mem,read}$ and $t_{mem,write}$ to the FR and FS, with small RMSE are taken. These models are given below:

$$t_{mem,active} = 0.07*FR*FS + 0.008*FR + 0.063*FS + 0.013$$
$$t_{mem,read} = 0.02*FR*FS + 0.006*FR + 0.03*FS + 0.005$$

$t_{mem,write} = 0.02*FR*FS + 0.0006*FR + 0.02*FS + 0.002$
$t_{mem,idle} = 1 - t_{mem,active}$

Substituting the above equations in the memory power model (Equation (44)) gives:

$$P_{mem,T} = 136.4*FR*FS + 12.0*FR + 144.5*FS + 481.4 \quad (45)$$

The Root Mean Square Error of the above model is calculated to be 17mW, which is shown in the table below:

	FR*FS	FR(fps)	FS	$P_{mem,T}$ (actual) (mW)	$P_{mem,T}$ (predicted) (mW)
1	1	1	1	802	774
2	0.25	1	0.25	575	564
3	0.06	1	0.06	517	510
4	0.83	0.83	1	775	749
5	0.21	0.83	0.25	565	556
6	0.05	0.83	0.06	510	507
7	0.42	0.42	1	715	688
8	0.1	0.42	0.25	540	536
9	0.03	0.42	0.06	490	499
		Root Mean Square Error			17mW

Table 29: Measured and compositional model predicted values for $P_{mem,T}$ and the corresponding RMSE

6.6 Comparison of the white box and black box models (without CPU power down)

This section compares the models obtained from the white box and black box approaches. The compositional model of the white box approach from Section 6.5 and the black box model from the Section 6.2.1 are given below:

$$P_{mem,T} = 136.4*FR*FS + 12.0*FR + 144.5*FS + 481.4$$

$$P_{mem,T} = 110.5*FR*FS + 37.3*FR + 191.4*FS + 462.3$$

The RMSE of the white box model is 17mW (refer Table (29)) which is 11 times larger than the RMSE of the black box model (1.55mW from Table (17)). Therefore, the black box model is more accurate than the white box models. The reason for large RMSE of the white box model is obvious from the method of composition of the models, in which the errors of individual models add up.

From both the models (white box and black box models), it can be observed that the term FS (number of pixels per frame) has large influence on the power consumption when compared to the terms; FR (number of frames per second) and FR*FS (number of pixels per second). As described in Section 6.2.2.1, the large influence of FS term on the power consumption of the memory suggests that the hardware blocks (specially MBS block) have more influence on the power consumption of memory than the CPU. Another observation from the models is that there is a large amount of constant offset power (481.4mW and 462.mW) consumed by the memory, independent of the application parameters. This offset power is due to the clock power of the logic when the memory is in idle state.

7 Integrated power model from the power models of SoC and memory

7.1 Introduction

Chapters 5 presented a black box model that relates the average power consumption of SoC to the application parameters. Chapter 6 presented a black box model that relates the average power consumption of memory to the application parameters. Since SoC and DDR memory are two independent components and the power measurements were done for both the components separately, we can combine the power models for SoC and memory to obtain an integrated model. Through the integrated model, we get a high-level model that predicts the power consumption of the MPEG-4 decoder application from the application parameters.

The integrated model can also be developed for the white box models. But, we chose black box models because these models are more accurate than the white box models with small RMSE (refer Section 5.8 and 6.6). We chose CPU power down mode models to make the integrated model, because the average power consumption with CPU power down mode is smaller than that of without CPU power down mode.

7.2 Integrated power model

From the Table (15) and Table (19), the normalized models for predicting the average power consumption of the SoC ($P_{SoC,T}$) and memory ($P_{mem,T}$) with small RMSE (2.15mW and 2.34mW respectively) are chosen to make the integrated model.

$P_{SoC,T}$ = 106.5*FR*FS + 7.9*FR +14.7*FS+ 536.3
$P_{mem,T}$ = 99.7*FR*FS + 37.9*FR +192.9*FS+ 473.4

The above two models for predicting the average power consumption of the SoC and memory are in the same format, therefore, combining them gives an integrated model that predicts the net average power consumption from the application parameters.

$$P_{net} = P_{SoC,T} + P_{mem,T} \qquad (46)$$

$$P_{net} = 206.2*FR*FS + 45.8*FR + 207.6*FS + 1009.7 \qquad (47)$$

The RMSE of the above model is the sum of RMSEs of the individual SoC and memory power models i.e. 4.5mW

7.2.1 Analysis

From the coefficients of the Equation (47), it can be observed that the terms FR*FS (number of pixels per second) and FS (number of pixels per frame) have the same influence on the net average power consumption.

When we observe the power models of SoC and memory individually, the term FR*FS has large influence on the power consumption of SoC where as the term FS (number of pixels per frame) has large influence on the power consumption of memory (refer the Section 6.2.2.1 for explanation). From the Section 6.2.2.1, we observed that on SoC side CPU has more influence on the power consumption of SoC and on memory side hardware blocks have more influence on the power consumption of memory. But, when we combine the models both the terms (FR*FS and FS) got equal significance and therefore both CPU and hardware blocks have the equal influence on the net average power consumption.

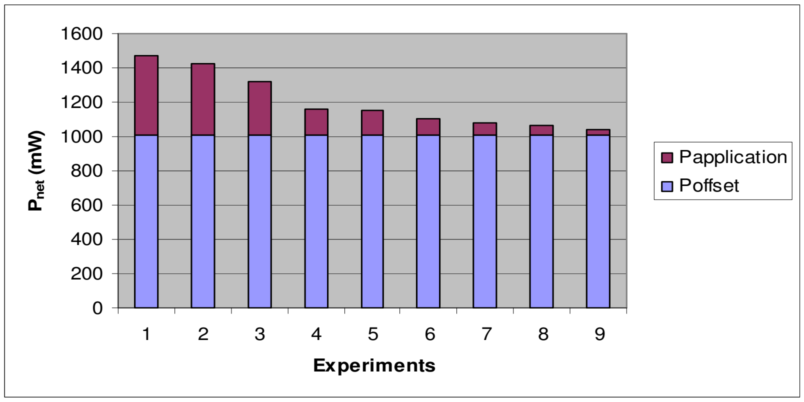
The term FR (number of frames per second) has less influence on the net average power consumption when compared to the other terms, because the hardware components QVCP and MBS operate at the output frame rate i.e.50Hz and are independent of the input frame rate (refer Section 5.4.1).

In Equation (47), the net average power consumption is influenced by two parts. One part is the constant offset power (P_{offset}) which is independent of the application parameters. The offset power is obtained by the clock power of the components during their idle periods. The other part is the power ($P_{application}$) that varies with the application parameters and is predicted by the model. The following table shows the P_{net}, P_{offset} and $P_{application}$ values for each experiment with different frame rate and relative frame size of the input stream. $P_{application}$ is calculated as the difference of P_{net} and P_{offset} values.

Experiment	FR(fps)	FS	P_{net} (mW)	P_{offset} (mW)	P_{net} - P_{offset} = $P_{application}$ (mW)
1	30	16	1471.6	1009.7	461.9
2	30	4	1421.4	1009.7	411.7
3	30	1	1322.5	1009.7	312.8
4	25	16	1160.7	1009.7	151.0
5	25	4	1148.7	1009.7	139.0
6	25	1	1106.6	1009.7	96.9
7	12.5	16	1079.0	1009.7	69.3
8	12.5	4	1064.9	1009.7	55.2
9	12.5	1	1043.4	1009.7	33.7

Table 30: P_{offset} and $P_{application}$ values calculated for different experiments

In Graph (23), the X-axis shows the experiment number and the Y-axis shows the net average power consumption. The graph shows the contribution of P_{offset} and $P_{application}$ to the net average power consumption (P_{net}).



Graph 23: Graph showing the contribution of P_{offset} and $P_{application}$ to the net average power consumption (P_{net})

The reduction in the net average power consumption by varying the application parameters frame rate and frame size from 30fps, 4cif resolution (Experiment 1) to 12.5fps, qcif resolution (Experiment 9) is 30%. From Graph (23), offset power (P_{offset}) is clearly the dominating part in the net average power consumption (P_{net}). The offset power is 85% (averaged over the experiments) of the net average power consumption. The offset power is obtained by the clock power of the components during their idle periods. In CPU power down mode, during the idle state CPU is clock gated. Therefore, the contributors to the offset power are the hardware blocks of the SoC and the DDR memory. The main contributors are the QVCP, MBS, control bus DCS, data bus PMAN, MMI and the DDR memory, which can not be clock gated during their idle periods in this platform. To reduce the energy consumption during idle periods of the components other than clock gating the components, dynamic frequency and voltage scaling can also be used. Chapter 9 discusses about the dynamic frequency and voltage scaling.

8 SoC and Memory experiments with a different input stream

8.1 Introduction

This chapter discusses the experiments performed with the same decoder application for a different input stream. These experiments were done to check the influence of content of the input stream on power models of the SoC and memory. Stream 1 (used for the experiments in earlier chapters) and Stream 2 (used for the experiments in this chapter) are completely different in their content. The streams are selected in such a way that we test the extreme conditions of motion in pictures. The Stream 1 is a slow motion picture where as the Stream 2 is a fast motion picture. If the streams with extreme motion conditions result in similar power consumption and internal parameters, then it strongly suggests that the models developed in the previous chapters can be used, in general, for any other input stream.

8.2 Experiments and results

The experiments were performed with CPU power down mode. Since, the earlier experiments were done at a CPU frequency of 100.5 MHz and Memory frequency of 199.8 MHz, the experiments in this chapter were also done at the same frequencies.

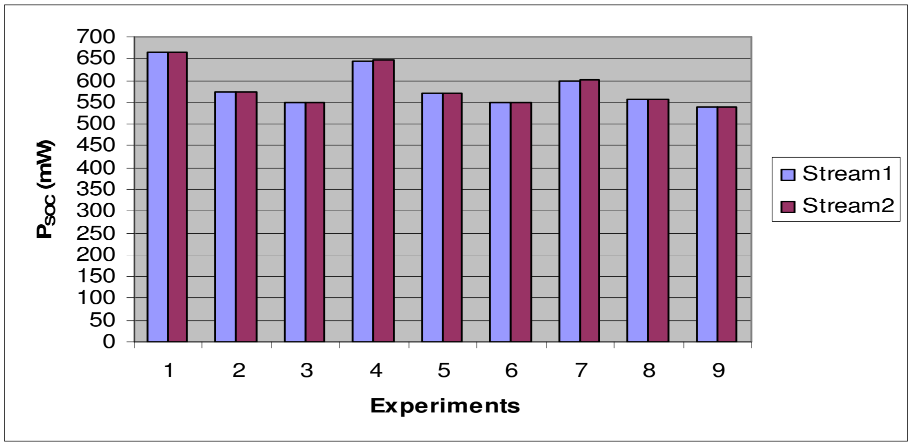
8.2.1 SoC experiments

The average power consumption and the application dependent platform parameters of the SoC are measured at three different frame rates and frame sizes of the input stream. The following table shows the measured values.

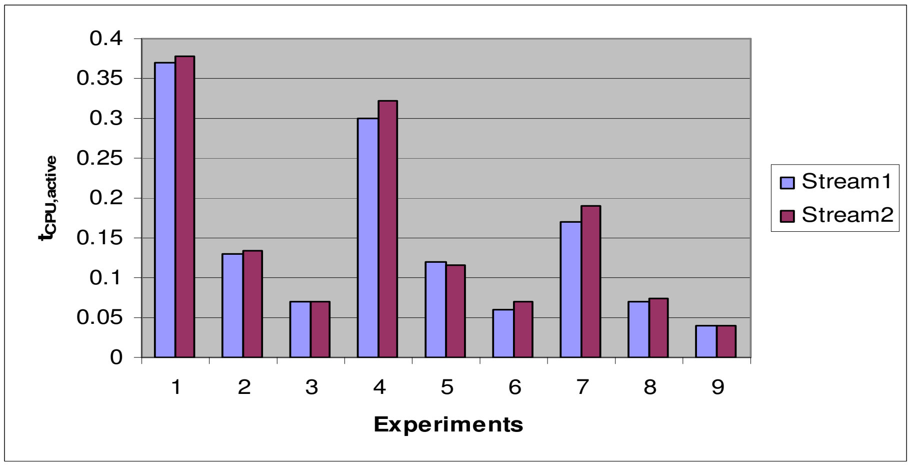
Experiment	FR(fps)	FS	$t_{CPU,active}$	$t_{CPU,stall}$	$t_{CPU,idle}$	$t_{QVCP,active}$	$t_{MBS,active}$	$P_{SoC,T}$(mW)
1	30	4cif	0.38	0.34	0.28	0.43	0.49	666.6
2	30	cif	0.14	0.15	0.72	0.43	0.42	575.7
3	30	qcif	0.07	0.10	0.83	0.43	0.42	550.5
4	25	4cif	0.32	0.29	0.39	0.43	0.49	648.9
5	25	cif	0.12	0.13	0.75	0.43	0.42	570.6
6	25	qcif	0.07	0.10	0.83	0.43	0.42	547.9
7	12.5	4cif	0.19	0.18	0.63	0.43	0.49	603.5
8	12.5	cif	0.07	0.10	0.84	0.43	0.42	558.1
9	12.5	qcif	0.04	0.07	0.89	0.43	0.42	540.4

Table 31: The $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$, $t_{CPU,idle}$, $t_{QVCP,active}$ and $t_{MBS,active}$ values measured for Stream 2

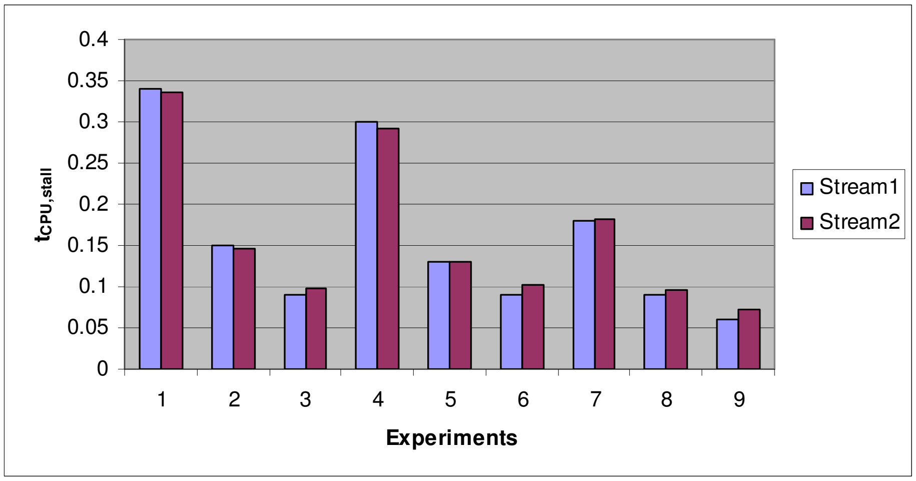
The values for $P_{SoC,T}$, $t_{CPU,active}$, $t_{CPU,stall}$, $t_{QVCP,active}$ and $t_{MBS,active}$ in the above table are compared to the values in Table (6) of chapter 5. The following graphs show the comparison. In the graphs, X-axis gives the number of the experiment in the order given in Table (31).



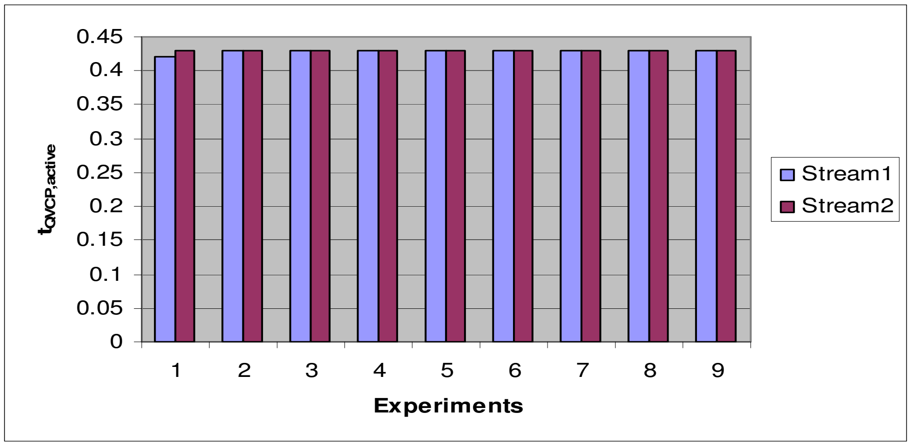
Graph 24: Graph comparing the $P_{SoC,T}$ values measured with stream 1 and stream 2



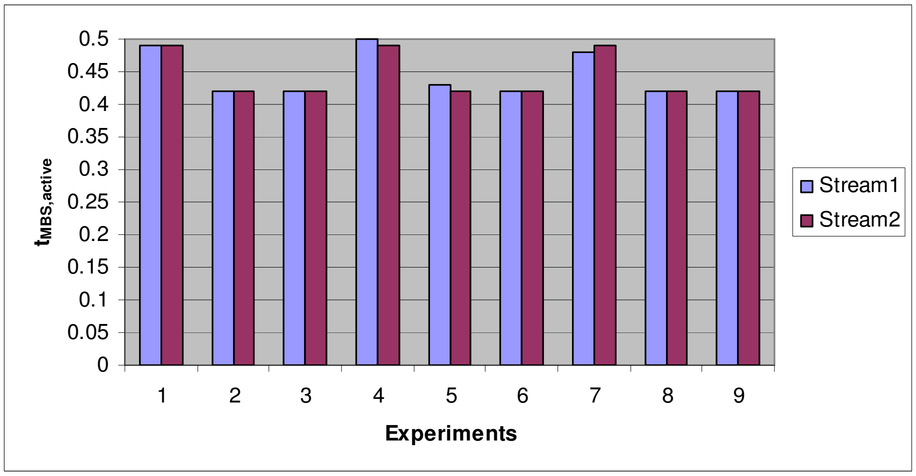
Graph 25: Graph comparing the $t_{CPU,active}$ values measured with stream 1 and stream 2



Graph 26: Graph comparing the $t_{CPU,stall}$ values measured with stream 1 and stream 2



Graph 27: Graph comparing the $t_{QVCP,active}$ values measured with stream 1 and stream 2



Graph 28: Graph comparing the $t_{MBS,active}$ values measured with stream 1 and stream 2

In half of the experiments, the parameters P_{SOC}, $t_{CPU,active}$, $t_{CPU,stall}$, $t_{QVCP,active}$ and $t_{MBS,active}$ are found to be the same for both the streams. The maximum variation for each parameter between Stream 1 and Stream 2 is given in the table below. There is no significant trend of increase or decrease in the parameter values between the streams.

Parameter	Max variation
$P_{SoC,T}$	0.85%
$t_{CPU,active}$	14.2%
$t_{CPU,stall}$	14.3%
$t_{QVCP,active}$	2.38%
$t_{MBS,active}$	2.32%

Table 32: The maximum variation of the Stream 2 parameters with respect to the Stream 1 parameters

The maximum variation percentage for the $t_{CPU,active}$ and $t_{CPU,stall}$ in the above table seems large but actually they occurred for very small values of $t_{CPU,active}$ and $t_{CPU,stall}$.

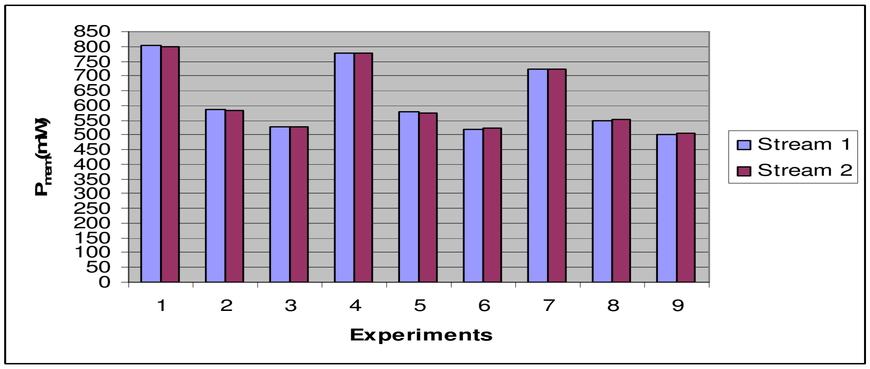
8.2.2 Memory experiments

Similarly, the average power consumption and the Set 2 parameters of the memory are measured for the nine different combinations of frame rate and frame size. The following table shows the measured values.

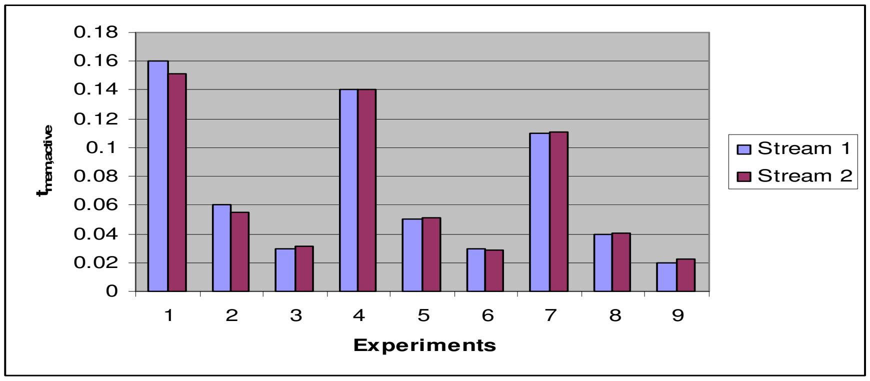
Experiment	FR (fps)	FS	$t_{mem,idle}$	$t_{mem,active}$	$t_{mem,read}$	$t_{mem,write}$	$P_{mem,T(measured)}$ (mW)
1	30	4cif	0.85	0.15	0.064	0.050	800
2	30	cif	0.94	0.06	0.016	0.013	584
3	30	qcif	0.97	0.03	0.005	0.003	528
4	25	4cif	0.86	0.14	0.064	0.050	778.5
5	25	cif	0.95	0.05	0.016	0.013	575
6	25	qcif	0.97	0.03	0.005	0.003	523
7	12.5	4cif	0.89	0.11	0.064	0.050	721.5
8	12.5	cif	0.96	0.04	0.016	0.013	552
9	12.5	qcif	0.98	0.02	0.005	0.003	506.5

Table 33: The $P_{mem,T}$, $t_{mem,idle}$, $t_{mem,active}$, $t_{mem,read}$ and $t_{mem,write}$ values measured for Stream2

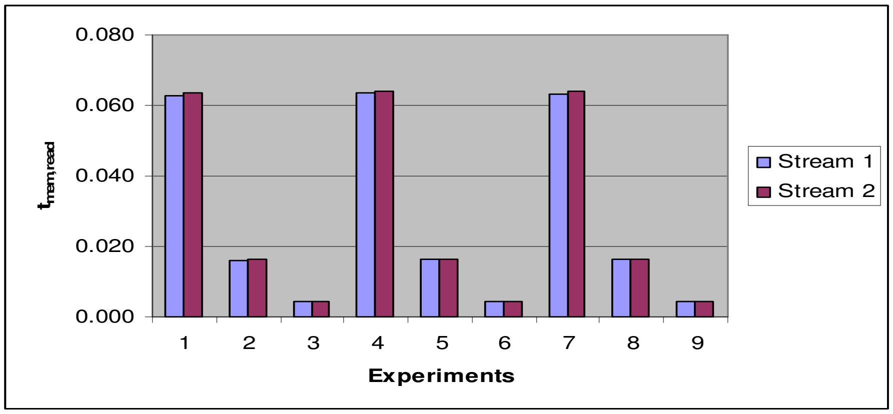
The values measured for the $P_{mem,T}$, $t_{mem,active}$, $t_{mem,read}$, $t_{mem,write}$ for stream1(refer Table (21)) are compared with the values measured for stream 2 (refer Table (33)). The following graphs show the comparison.



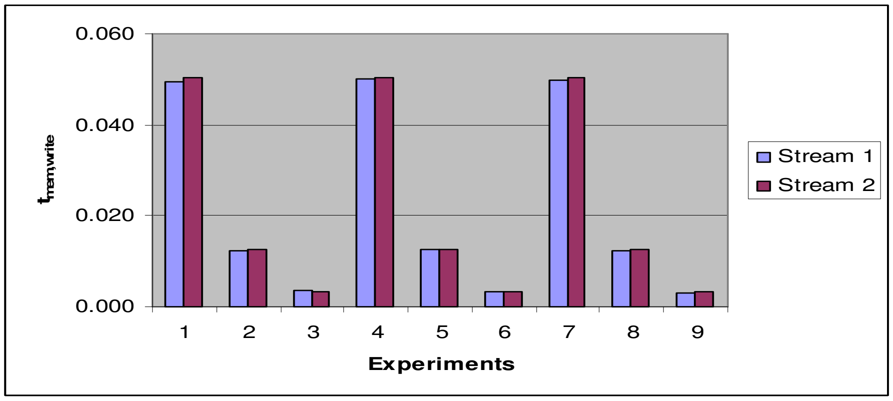
Graph 29: Graph comparing the $P_{mem,T}$ values measured with stream 1 and stream 2



Graph 30: Graph comparing the $t_{mem,active}$ values measured with stream 1 and stream 2



Graph 31: Graph comparing the $t_{mem,read}$ values measured with stream 1 and stream 2



Graph 32: Graph comparing the $t_{mem,write}$ values measured with stream 1 and stream 2

In case of memory also, for half of the experiments, the parameters $P_{mem,T}$, $t_{mem,active}$, $t_{mem,read}$ and $t_{mem,write}$ are found to be the same for both the streams. The maximum variation for each parameter between Stream 1 and Stream 2 is given in the table below. There is no significant trend of increase or decrease in the parameter values between the streams. Therefore, the variations seem to be more of a measurement error rather than a trend.

Parameter	Max variation
$P_{mem,T}$	1.16%
$t_{mem,active}$	6.25%
$t_{mem,read}$	25%
$t_{mem,write}$	25%

Table 34: The maximum variation of the Stream 2 parameters with respect to the Stream 1 parameters

The maximum variation percentage for the $t_{mem,read}$ and $t_{mem,write}$ in the above table seems large but actually they occurred for very small values of $t_{mem,read}$ and $t_{mem,write.}$

8.3 Conclusion

From the experiments of this chapter, we observed that there is no significant variation of the power consumption and application dependent platform parameters of the SoC and memory for input streams with different content. These results strongly suggest that the content of the input stream has no influence on the power models of the SoC and memory.

9 Effect of frequency scaling

9.1 Introduction

The compositional model given in chapter 7 was developed by considering fixed frequency and voltage method. All the experiments in previous chapters were done at a fixed frequency of CPU i.e. at 100.5MHz. At this frequency, CPU spends only 4% of the total time (refer Table (3)) in active state for decoding an input stream with 12.5fps and qcif resolution. In this case, if we scale down the frequency of the CPU such that CPU spends most of the time in active state, we can save the energy consumption in idle state. Figure (13) shows the two cases, without frequency scaling and with frequency scaling.

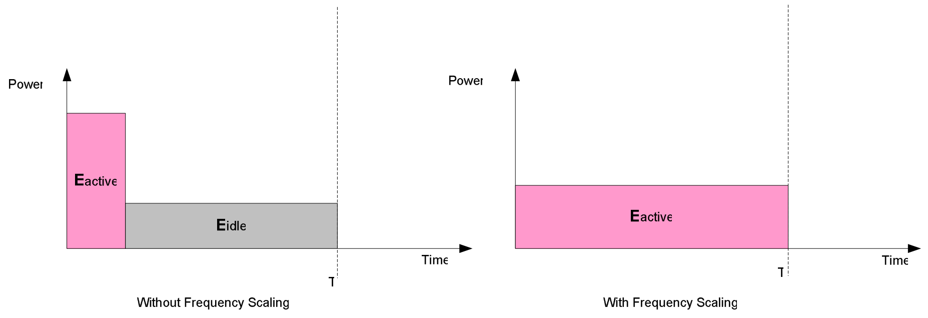


Figure 13: Energy consumption without and with frequency scaling

Energy consumption in active state remains same for both the cases, because when we scale down the frequency active time increases linearly but active power decreases linearly (refer Equation (1) of chapter 4). Frequency scaling gives only linear reduction in the energy consumption, but by reducing frequency we can also reduce the supply voltage which gives quadratic reduction in the energy consumption. We can also save the energy consumption in idle state, by using clock gating technique in which the clock of the CPU is disabled during idle state. This technique was used in the experiments of previous chapters and referred as CPU power down mode (refer Section 5.3 of Chapter 5).

In this chapter, by using the application dependent platform parameters of the CPU, $t_{CPU,active}$ and $t_{CPU,stall}$ we calculate (using linear relation between time and frequency) scalable frequencies for the CPU for different frame rates and frame sizes of the input stream. With the calculated frequencies, we performed experiments to observe the effect of frequency scaling on power consumption. Frequency scaling does not give reduction in the average power consumption with CPU power down mode, because CPU is already clock gated during idle state in this method.

9.2 Frequency scaling of CPU

Table (35) shows the frequency at which CPU would be in active and stall states for 80% of the total time for each combination of FR and FS with CPU power down mode. We leave a margin of 20% to make sure that the timing constraints are not violated. Since frequency and time vary linearly with each other the following equation is taken to calculate the scalable frequencies.

$f_{scaled} = (t_{orig}/0.8) * f_{orig}$

f_{scaled}: The scaled frequency at which CPU is in active and stall states for 80% of the time
t_{orig}: Percentage of time CPU is in active and stall states at 100.5MHz frequency
f_{orig}: Reference frequency of the CPU i.e.100.5MHz

t_{orig} value is calculated as the sum of $t_{CPU,active}$ and $t_{CPU, stall}$ values in the Table(3) of Chapter 5.

	FR(fps)	FS	f_{orig}	t_{orig}	f_{scaled}
1	30	4cif	100.5	71	89
2	30	cif	100.5	28	35
3	30	qcif	100.5	16	20
4	25	4cif	100.5	60	75
5	25	cif	100.5	25	31
6	25	qcif	100.5	15	19
7	12.5	4cif	100.5	35	44
8	12.5	cif	100.5	16	20
9	12.5	qcif	100.5	10	13

Table 35: Scalable frequencies

By using the estimation of f_{scaled} values from the above table, experiments are performed with nine combinations of frame rate and frame size of the input stream and the average power consumption across SoC (P_{SOC,T_f_scaled}) is measured for each combination (refer Table (36)). It is not possible to set exact values for f_{scaled} as given in the Table (35), because of the PLL settings. Therefore in the experiments, f_{scaled} is selected to be close to the theoretical values given in Table (35).

	FR (fps)	FS	t_{orig} (%)	f_{orig} (%)	t_{scaled} (%)	f_{scaled} (MHz)	P_{SoC,T_orig} (mW)	P_{SoC,T_f_scaled} (mW)
1	30	4cif	71	100.5	77	99.9	666.6	685.4
2	30	cif	28	100.5	73	37.5	575.7	583.0
3	30	qcif	16	100.5	78	20.3	550.4	562.9
4	25	4cif	60	100.5	83	75	643.9	661.9
5	25	cif	25	100.5	79	30.4	570.6	586.4
6	25	qcif	15	100.5	75	18.8	547.9	560.3
7	12.5	4cif	35	100.5	84	40.5	598.4	615.0
8	12.5	cif	16	100.5	74	20.3	558.1	568.1
9	12.5	qcif	10	100.5	87	10.1	540.4	534.2

Table 36: SoC power consumption measured with frequency scaling for CPU power down mode

As described earlier the average power consumption of the SoC with frequency scaling (P_{SOC,T_f_scaled}) is not reduced when compared to the average power consumption of the SoC with CPU power down mode (P_{SOC,T_orig}). The reason for increase of P_{SOC,T_f_scaled} value when compared to P_{SOC,T_orig} is that the relation between frequency and power consumption is not exactly linear. This is because, the power consumption of SoC ($P_{SoC,T}$) also includes leakage power (Equation (1) of Chapter 4) which remains constant with frequency changes and only varies with voltage.

But, without power down mode, we can observe significant reduction in the average power consumption with frequency scaling (refer Table (37)). Table (37) shows the average power consumption of the SoC with frequency scaling (P_{SOC,T_f_scaled}) for an input stream with more number of pixels per second (30fps and 4cif resolution) and less number of pixels per second (12.5fps and qcif resolution). For an input stream of 12.5fps and qcif resolution P_{SOC,T_f_scaled} value at 10.1 MHz, is reduced by 21% when compared to P_{SOC,T_orig} value at 100.5MHz. According to the linear relation between frequency and power, in this case we expect for a 90% of reduction in power (from 100.5MHz to 10.1MHz: 90% reduction in frequency). But actually the power is reduced only 21%.

As described earlier, here the relation between frequency and power consumption is not exactly linear because of the leakage power (Equation (1) of Chapter 4) which remains constant with frequency changes and only varies with voltage. The P_{SOC,T_f_scaled} value for input stream with 12.5fps and qcif resolution is reduced by 25.3% with reference to the input stream with 30fps and 4cif resolution. Without frequency scaling the reduction is only 6% (refer Table (37)).

	FR (fps)	FS	t_{orig} (%)	f_{orig} (%)	t_{scaled} (%)	f_{scaled} (MHz)	P_{SoC,T_orig} (mW)	P_{SoC,T_f_scaled} (mW)
1	30	4cif	88	100.5	89	99.9	717.1	715.2
2	12.5	qcif	23	100.5	75	10.1	674.2	534.2

Table 37: SoC power consumption measured with frequency scaling for without CPU power down mode

9.3 Conclusion

This chapter shows that, by using the application dependent platform parameters of white box approach we can estimate the scalable frequencies for the CPU. Therefore, the application dependent platform parameters can be used to actively control the CPU frequency of operation.

10 Conclusions

In this thesis, the power models for the SoC and memory are developed with two different approaches namely black box and white box approaches. The black box models relate the power consumption of the SoC and memory to the application parameters without considering architecture level details. The black box models are abstract and easy to model. The white box approach models relate the power consumption of the SoC and memory to the application parameters, by considering the architecture level details and through composition of the models. Because of the method of composition, the RMSE of the white box models is larger than that of the black box models. Therefore, the black box models are more accurate than the white box models. But the validity of the black box models is limited to this specific platform and application.

Regardless of the accuracy of the white box models, there are several advantages of this approach. This approach analyses the platform parameters that cause the power consumption in detail. The platform parameters measured in this approach can be used as an estimation for other platforms and applications. As discussed in Chapter 9, the application dependent platform parameters can be used to actively control the CPU frequency of operation and thereby the power consumption. Moreover, the application dependent platform parameters can be used to investigate the compositionality in applications. For example, for running two applications on a CPU concurrently, we need to have an estimate of the CPU utilization by these applications. If we know the time spent by CPU in active and idle states for two applications separately, this information can be used to estimate the timing requirements for the compositional application. As a continuation of this work, it is recommended to investigate the compositionality in multimedia applications by using the application dependent platform parameters of the applications.

Besides the individual models for the power consumption of SoC and memory, this thesis also presents the net power consumption model by integrating the individual models of SoC and memory. From the SoC power model, the observation was that the CPU has more influence on the power consumption of SoC than the hardware components. From the memory power model, it was observed that the hardware components have more influence on the power consumption of memory than the CPU. When we integrated the individual models of SOC and memory we observed that both CPU and hardware components have equal influence on the net average power consumption. From the integrated model we also observed that the offset power (clock power of the components during their idle periods) is the dominating part in the net average power consumption. The offset power is 85% (averaged over the experiments) of the net average power consumption.

In this thesis, the experiments were performed with the same decoder application but with two different input streams whose content represents the extreme conditions of motion in the pictures. The results from the experiments suggest that the content of the input stream has no impact on the power consumption and on the application dependent platform parameters. But, it is necessary to perform experiments with some more input streams to make conclusions about the influence of the input stream content on the power models.

Validating the white box approach models through experiments in the PNX1500 platform involves quite a lot of effort. This platform does not allow the measurement of all the parameters needed by the models, for example the time required for the burst activity ($t_{mem,burst}$) in the memory model. It is necessary to have platforms that support performance measurements by providing required performance counters and registers.

In this thesis, the power models were developed by using linear regression method. Accuracy of results obtained from linear regression method depends on, how well the parameters are independent from one another. One technique to make the parameters independent is to run special test programs that characterize each parameter separately. But, if the parameters are strongly correlated and it is not possible to have special test programs, then combining the parameters gives more accurate results than separating them.

References

[1] Design for Low-Power at the Electronic System Level: http://www.soccentral.com/soccontent/documents/ESL_Design_for_Low_Power_ChipVision.pdf#search=%22Design%20for%20Low-Power%20at%20the%20Electronic%20System%20Level%22

[2] Investigating Hardware and Software Approaches to Increasing the Battery Life of mobile Embedded Systems http://www.kudurshian.net/projects/kudurshian3.pdf

[3] Philips, PNX1500 series data book 19 April 2005

[4] Philips, NDK Tools support, user manual v4.3, 18 April 2005

[5] Philips, NDK Software architecture, user manual v4.3, 18 April 2005

[6] Model 2700 Multimeter/Switch System User's Manual

[7] On-Chip Power Management utilizing an embedded hardware controller and a low-power serial interface: http://www.national.com/appinfo/power/files/OnchipPWRMgmtEmbeddedWorld021704.pdf

[8] Contribution to BETSY, version 0.4 by Liesbeth Steffens

[9] Calculating memory system power for *DDR2*, TN-47-04, Micron Technology, Inc., 2004.

[10] Linear regression : http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

[11] Philips, PSOS System Concepts

[12] Mathematica introduction: http://www.maths.uwa.edu.au/~fowkes/Courses/M3A5/Mathemintro.pdf#search=%22linearsolve%20mathematica%20%22

[13] A MATLAB guide to Linear Algebra: http://www.nyu.edu/classes/edwards/chap7.html

[14] Performance Measures for Numeric Predictions: http://grb.mnsu.edu/grbts/doc/manual/Error_Measurements.html

[15] Philips, PNX1500 series data book 17 March 2006

[16] Composition: http://mathworld.wolfram.com/Composition.html

[17] Philips, Getting started with PNX1500 software development, user manual v4.3, 18 April 2005

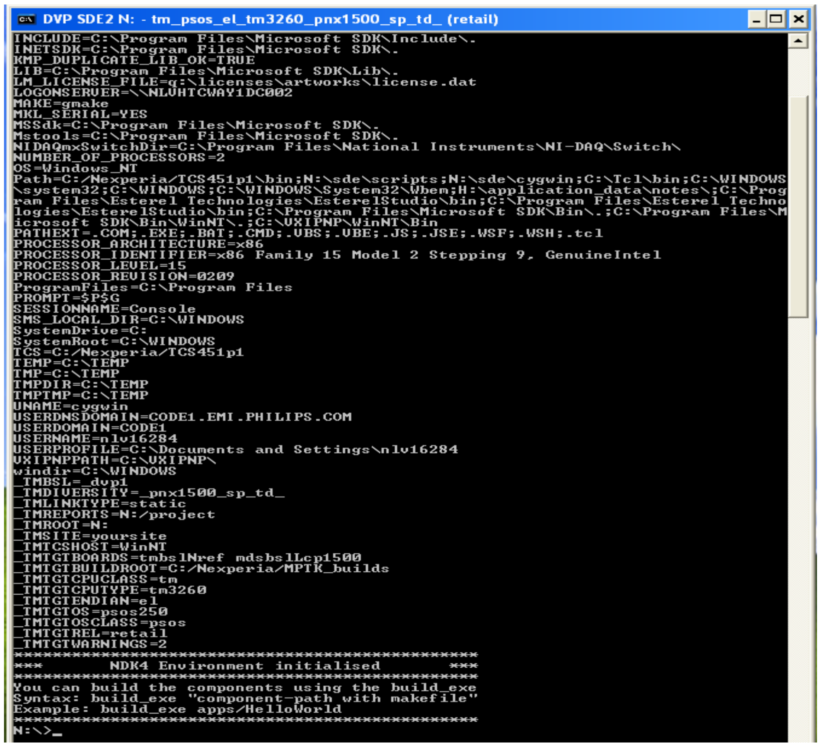
APPENDIX

❖ *Building programs to PNX1500 platform*

While building the programs to PNX1500, there are several environmental variables that need to be set [17]. NDK package is provided with a batch file named **ndk4_env.bat,** to set the environment variables. Target for which the programs need to be built was chosen as “***pnx1500 tm3260”*** in the batch file. Some of the other important variables used in this project are given below.

1. Build “flavour” can be Debug, Trace, Assert or Retail. ***Debug*** mode is needed to use Debugger. Because of the compiler option settings, code compiled in this mode runs at half the speed of code compiled at ***Retail*** mode. ***Trace*** mode is similar to that of ***Debug*** mode except the difference in compiler option settings. In ***Assert*** mode debug is not enabled but assertions are enabled. Assertions are used to check for programming errors. In ***Retail*** mode all assertions and traces are disabled at compile time. In this project ***Retail*** mode is chosen.
2. Host type: For a PCI plug-in board hosted operation it has to be set as WinNT.
3. Diversity: The diversity “ ***_sp_”*** is set to specify single processor mode. In order to enable TimeDoctor tool support diversity “***_td_”*** has been set.
4. Endianess: Is chosen as little endian ***“el”***. Even though PNX1500 CPU chip hardware is theoretically support big endian operation, the NDK/MPTK software is neither tested for the big endian mode nor supports it.

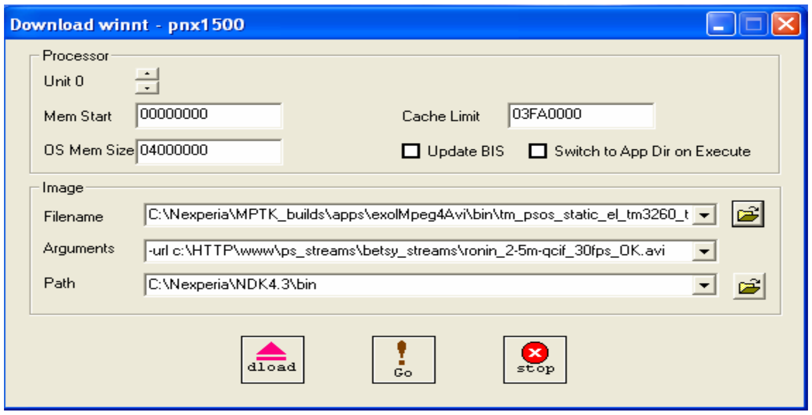
Figure (14) is a screen shot of **ndk4_env.bat,** from which all the settings of environmental variables described above can be seen.



```
INCLUDE=C:\Program Files\Microsoft SDK\Include\.
INETSDK=C:\Program Files\Microsoft SDK\.
KMP_DUPLICATE_LIB_OK=TRUE
LIB=C:\Program Files\Microsoft SDK\Lib\.
LM_LICENSE_FILE=q:\licenses\artworks\license.dat
LOGONSERVER=\\NLVHTCWAY1DC002
MAKE=gmake
MKL_SERIAL=YES
MSSdk=C:\Program Files\Microsoft SDK\.
Mstools=C:\Program Files\Microsoft SDK\.
NIDAQmxSwitchDir=C:\Program Files\National Instruments\NI-DAQ\Switch\
NUMBER_OF_PROCESSORS=2
OS=Windows_NT
Path=C:/Nexperia/TCS451p1\bin;N:\sde\scripts;N:\sde\cygwin;C:\Tcl\bin;C:\WINDOWS
\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;H:\application_data\notes\;C:\Prog
ram Files\Esterel Technologies\EsterelStudio\bin;C:\Program Files\Esterel Techno
logies\EsterelStudio\bin;C:\Program Files\Microsoft SDK\Bin\.;C:\Program Files\M
icrosoft SDK\Bin\WinNT\.;C:\VXIPNP\WinNT\Bin
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.tcl
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 15 Model 2 Stepping 9, GenuineIntel
PROCESSOR_LEVEL=15
PROCESSOR_REVISION=0209
ProgramFiles=C:\Program Files
PROMPT=$P$G
SESSIONNAME=Console
SMS_LOCAL_DIR=C:\WINDOWS
SystemDrive=C:
SystemRoot=C:\WINDOWS
TCS=C:/Nexperia/TCS451p1
TEMP=C:\TEMP
TMP=C:\TEMP
TMPDIR=C:\TEMP
TMPTMP=C:\TEMP
UNAME=cygwin
USERDNSDOMAIN=CODE1.EMI.PHILIPS.COM
USERDOMAIN=CODE1
USERNAME=nlv16284
USERPROFILE=C:\Documents and Settings\nlv16284
VXIPNPPATH=C:\VXIPNP\
windir=C:\WINDOWS
_TMBSL=_dvp1
_TMDIVERSITY=_pnx1500_sp_td_
_TMLINKTYPE=static
_TMREPORTS=N:/project
_TMROOT=N:
_TMSITE=yoursite
_TMTCSHOST=WinNT
_TMTGTBOARDS=tmbslNref mdsbslLcp1500
_TMTGTBUILDROOT=C:/Nexperia/MPTK_builds
_TMTGTCPUCLASS=tm
_TMTGTCPUTYPE=tm3260
_TMTGTENDIAN=el
_TMTGTOS=psos250
_TMTGTOSCLASS=psos
_TMTGTREL=retail
_TMTGTWARNINGS=2
************************************************
***       NDK4 Environment initialised       ***
************************************************
You can build the components using the build_exe
Syntax: build_exe "component-path with makefile"
Example: build_exe apps/HelloWorld
************************************************
N:\>_
```

Figure 14: Screen shot of ndk4_env.bat

In dvpMon, PCI channel option is chosen to enable communication through PCI. Figure (15) is a screen shot of dload.exe, which is a dvpMon's command line interface to download applications to TriMedia. For decoder application .out file along with encoded stream is downloaded to TriMedia.


Download winnt - pnx1500

Processor
Unit 0
Mem Start 00000000
Cache Limit 03FA0000
OS Mem Size 04000000
☐ Update BIS ☐ Switch to App Dir on Execute

Image
Filename C:\Nexperia\MPTK_builds\apps\exolMpeg4Avi\bin\tm_psos_static_el_tm3260_t
Arguments -url c:\HTTP\www\ps_streams\betsy_streams\ronin_2-5m-qcif_30fps_OK.avi
Path C:\Nexperia\NDK4.3\bin

Figure 15: Screen shot of dload.exe

❖ *Multimeter*

This instrument has a 6½-digit display and can store up to 55,000 readings in the internal buffer. Figure (16) shows the front panel of the Multimeter. There are several keys on the front panel that help user to operate the multimeter. The function of some keys is described below.

- Store: Using this key it is possible to select number of readings to be stored during the execution of the application.
- Recall: This key is used to display stored readings and buffer statistics. It displays average voltage, standard deviation, minimum and maximum voltages measured.
- Rate: This key is used to set the integration time (measurement speed) of the A/D converter, i.e. the period of time input signal is measured.

It is also possible to remote program the multimeter through a Standard I/O Interface RS-232. National Instruments LabVIEW package is used to collect and analyze data stored in the buffer.

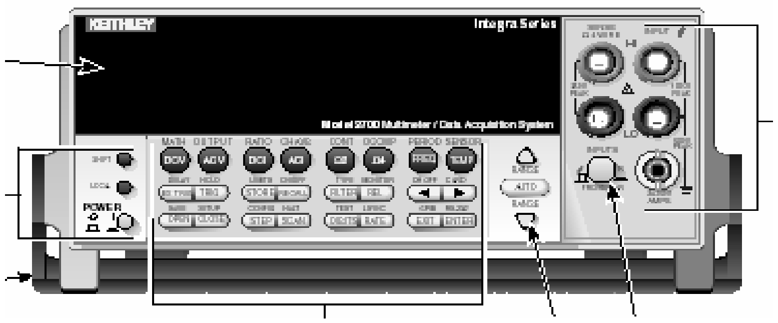


Figure 16: Front panel of the multimeter

❖ *Steps for an experiment*

In this section, overall steps for performing an experiment are described in order.

1. Build the application using ndk4_env.bat for target PNX1500
2. Reset the target using dvpMon or URD
3. Frequency of the CPU can be set to required value using URD registers. Default is 300.375MHz.
4. Program Keithly multimeter from PC through Keithly communicator for the required settings.
5. Use dload.exe of dvpMon, to download application in .out format along with encoded stream to target.
6. Start Keithly multimeter buffer to store the voltage readings.
7. Dump TimeDoctor buffer when the execution of application is finished.
8. Use LabView to collect and analyse the data stored in buffer