FLORIDA ATLANTIC UNIVERSITY

MOTOROLA OPP COMPONENT MODELING: ESTIMATION OF POWER CONSUMPTION AS A SYSTEM COST ON EMBEDDED DEVICES.

CAMILO CRUZ

July 21st, 2006

CONTENTS

INTRODUCTION

Power consumption has become a major issue in the design of consumer electronic devices as a result of increased demands for features and performance.

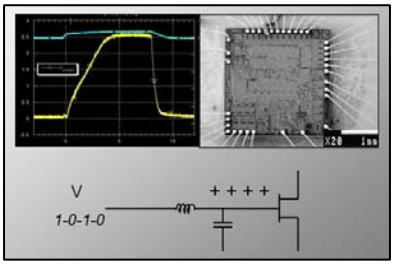
This trend affects the battery life of products, environmental issues due to larger required batteries and a higher product cost.

A component modeling approach allows us to approximate a given system's parameters in order to quantify the resources needed for a given application. In this particular project, under the direction of Drs. Ravi Shankar and Hari Kalva, we advance the modeling of power consumption as system cost, with an emphasis in the multimedia area of the application layer of the system design framework.

As systems implement more complex and higher quality video and sound, optimization of energy usage becomes essential for product feasibility. The current area of concern regarding energy usage in multimedia applications is CPU and RAM activity switching

POWER CONSUMPTION OVERVIEW

The main concern up to now regarding system consumption has been switching transitions in the address and data buses between CPU and RAM; this occurs since CMOS logic



works ideally as voltage controlled switches, with negligible gate current at the input. Thus, the main work or energy requirement is to move those static charges to and from the gates, to switch the transistors on or off. Also, when a CMOS pair is being switched (one mosfet goes on, the other off) there is a very brief short circuit condition when the two transistors have some simultaneous conduction, wasting energy away. Further, the intrinsic capacitance of the bus lines -even if they are short- implies that to raise or lower the gate to a potential (change of logic level to 1 or 0), the whole line has to be charged or discharged, making the energy needed even larger, considering that we have billions of individual bus line transitions in very shorts periods of time.

Ideally, a stable state consumes no additional energy, since the electrical charges stay on the transistor gates without a current flow. That is why so far we have aimed to quantify and optimize only the number of switching transitions in a given application.

It is important to note that emphasis is made to the switching occurring in the address and data buses, since they have an energy requirement ten to forty times greater than the internal CPU instruction switching. In this sense, it might be obvious to the reader that actual physical circuitry size has an important influence here. Here we can see some typical values of power usage for buses and CPU.

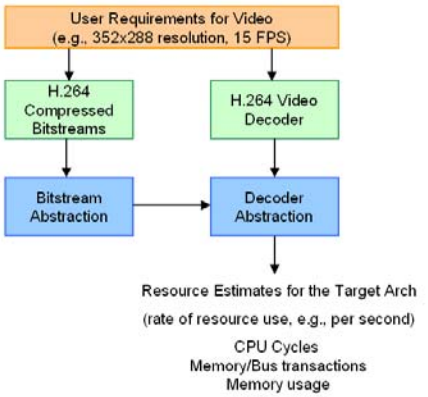
Table 1: Example of Energy Model

Parameter	Event		Energy(pJ)
$E_{instr(i)}$	Non Execution		100
	ALU (AND, EOR, etc.), MOV		250
	LDR		400
E_{CPU_idle}	Clock		50
$E_{memory\_acc(j,k)}$	Internal RAM	read	4400
		write	4300
	External	read	16800
		write	17200
$E_{bus\_acc(m,j)}$	Non-Sequential Access		500
	Sequential Access		300
$E_{bus\_idle\_clock(m)}$	-		100

MODELING APPROACH

The first thing we do to model our multimedia power consumption envelope is to break down and constrain the tasks to be analyzed. It is important to note that the audio portion of the multimedia analysis will be kept at a high level, based on average execution times, since its demand on system resources is estimated to be about only one tenth of the video portion.

We concentrate on the MPEG4 video standard, but also especially in the H264 standard, the newer codec highly suited to mobile applications, expected to be used in all new generation devices. We aim to start modeling with a given user requirement, like for example, a 352x288 video at 15 frames per second, and determine the resources needed for the target architecture.



Video playback on an embedded device can be shown to be dependent on two main areas: First, the actual content of the video image; its complexity (detail) as well as the amount of movement within, have both a direct impact on the amount and pattern of executed instructions, system transactions, and cache usage throughout rendering. We are currently working towards the abstraction of these parameters in order to be able to correlate them with system activity and have them be part of our modeling envelope. Secondly, the actual decoder software implementation has a great impact on system behavior (thus, bus transactions), depending on the way the programmers setup their subroutines and the different encoding/decoding parameters set for the videos.

Our modeling methodology currently includes the following steps and tools:

-Bitstream encoding, using Generic and Nokia Mpeg4 and H264 encoders

-Code image creation using Armulator

-Simulation, using Armulator

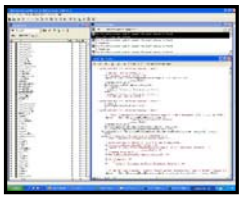
-Data migration and analysis, using Microsoft Excel

-Parametrization

-Hardware validation with Intel's Vtune Performance analyzer and Dell's Axim Pocket PC

In the bitstream generation step, we encode the test videos we want to model our decoder; in this case we use three standard video contents that reflect typical context seen on an embedded device. The process involves setting the different parameters for each test video, namely nine different bitrates of 15, 25, 50, 100, 128, 256, 384, 512, 740kBs, encoded to 15 fps and a resolution of 176x144 pixels.

These settings are set differently for each video standard; our MPEG4 requires the modification of control files, which are looked up by the encoder, while our H264 encoding requires command-line parameters (set by the use of DOS batch files).

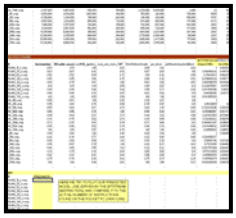


Once all the videos have been encoded, we have to compile an executable ‘image’ of the decoder software; that is, to create a version of the embedded decoder that can run under the ARM simulator, called AXD debugger -This simulated image is called an ‘AXD image’-. At this stage of development the code remains unmodified for all tests, except in that its command line input was disabled and replaced by an explicit order on what video file name to look for.

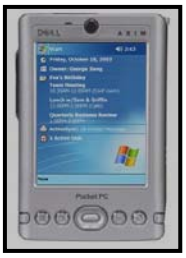
One particular detail was key in improving the speed of the simulations was by changing the compiled executable from a ‘debug’ version to a release version allowing the process to avoid much unnecessary annotation. That change alone brought the individual simulation time from ten hours on a P4 machine, down to under an hour for the same code.

Also, the settings currently available to run the ARM simulator are manually set on configuration files called ´peripherals.ami´ and ´tracer.ami´; On these, you can set whether to track the bus states, specific memory contents, etc.

When the AXD simulation (ARMULATOR) is run, it will create what is called a trace file, which is basically a text based file with a line by line write-down of the address and data buses contents at each Bus-cycle. Keep in mind that this data alone does not tell us the number of logic-level transitions nor the power requirements; we have to further analyze those numbers to start developing our model. In order to do this, we run ‘Power Monitor’, a program written by a previous engineer, which runs a line-by-line computation of the trace file. This basically counts the logic transitions in the simulated execution of the multimedia. Once we obtain all the numbers for all videos, they are input into Microsoft Excel for graphing and parametrization against bitstream abstraction patterns obtained from the encoding runs.



Of equal importance is the use of Dell’s Axim Pocket PC as a validation mechanism for our methodology, allowing us to obtain data from real ARM architecture hardware. This in order to correlate the simulated patterns with the ones from an actual Intel device.

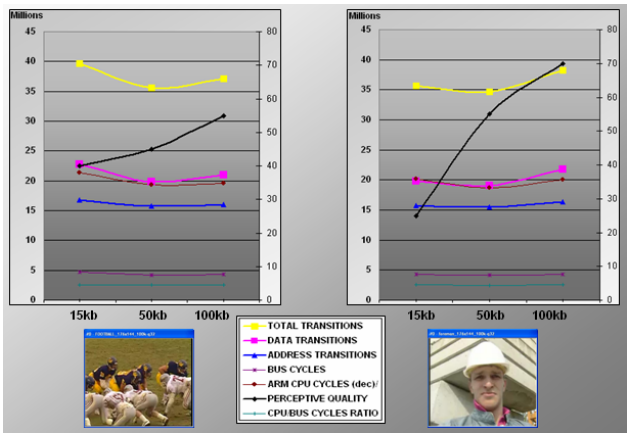


This has been achieved through the use of Intel's Vtune performance analyzer tool, which allows you to set up a tracing agent within the device. The agent will work on the background sampling the different processing events during the application run, in our case of course H264 decoding.

CURRENT RESULTS

- MPEG4:

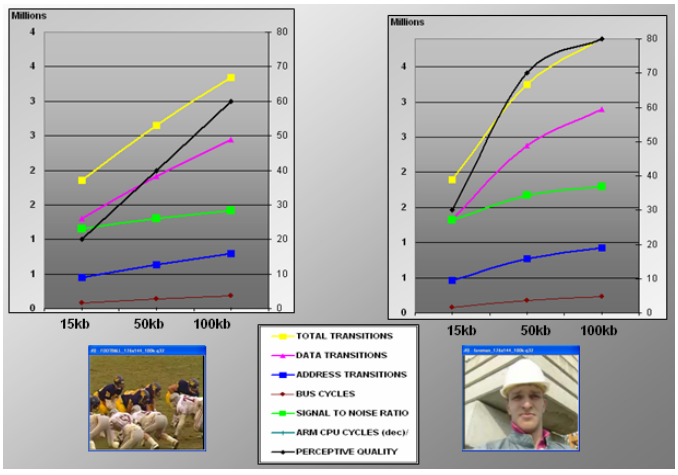
Some of the initial data we obtained for the MPEG4 shows the relationship between the different compression rates and the number of bus transitions, which, as stated before, are directly related to power consumption.



In general, the numbers reflect the to the left is in millions of transitions, which can later be correlated in our model to actual energy values in Joules, depending on the actual architecture configuration.

-H264

The lines for the H264 show a more lineal energy-wise behavior. In this particular type of study the architect could see for example how it might be advantageous in a certain application to upgrade the bandwidth from 15kb to 50kb to improve quality; but he may notice that going to 100kbs might not necessarily bring an appreciable quality improvement for the extra energy used.

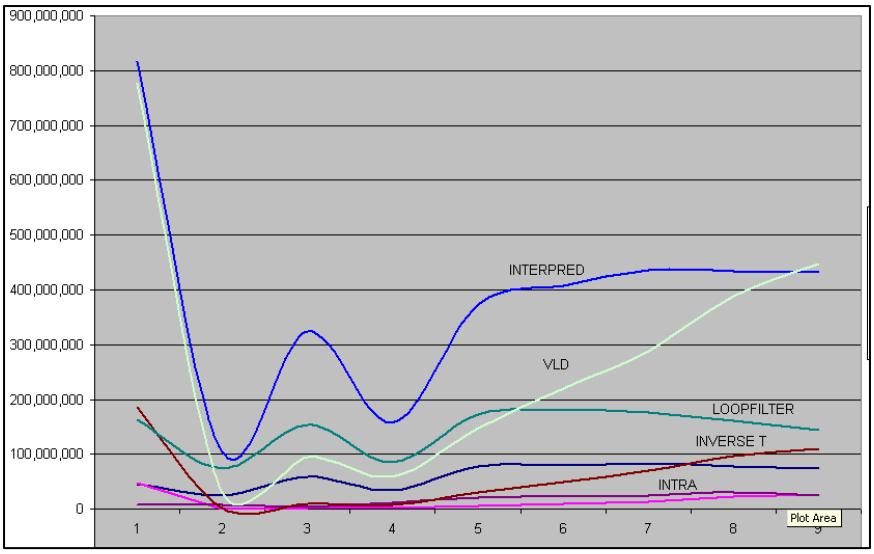


The above results still need to be parametrized against the video-content abstraction data in order to obtain a bitstream-driven model of the decoder(s). Careful work is necessary to overcome non-linearity and the different orders of magnitude (scale) of the various lines.

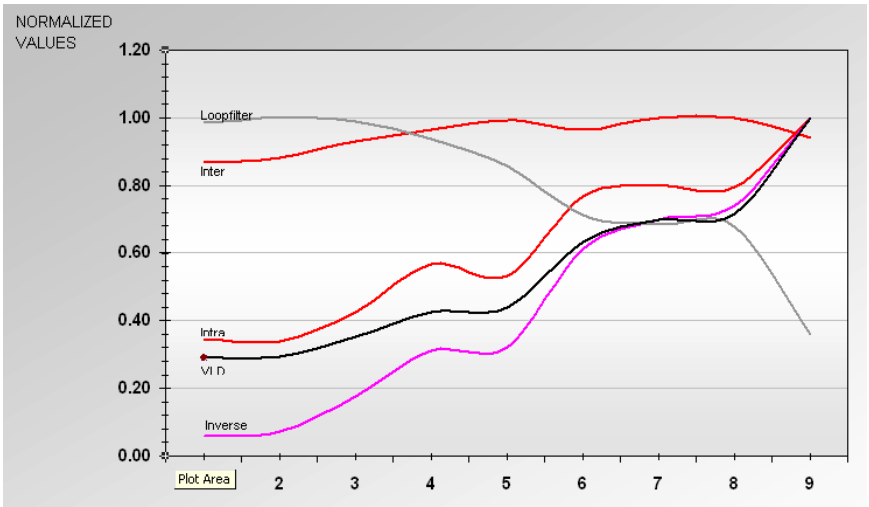
-Axim Pocket PC against Intel PC patterns

Some of the recently obtained data for the Axim Pocket PC runs of the decoder shows how the Vtune tool allows us to see the code behavior on a per-component basis.

Axim

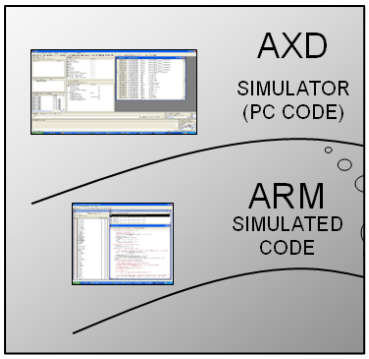


Pentium4



METHODOLOGY OPTIMIZATION/INTEGRATION

Some elements have been simultaneously advanced to accelerate the research and simulation process itself as an integral part of the OPP package. These include the addition of detailed step-by-step instructions for all stages of the game, allowing any upcoming engineer to get up to speed with OPP as quickly as possible. Also, so far an extensive organization of the file and directory system of the on-screen working environment has been performed in order to speed up every step of the process. .



Additionally, as part of this streamlining and integration aim, we are targeting the complete customization of the ARM simulation environment. We need to do it in a way that all data is gathered and analysed automaticaly without the need for intermediate trace files or additional applications, essentially turning the modeling process into a single-click operation. Fortunately, as one of our engineers has found (and described in his own white paper), the Eclipse development platform seems well suited for the kind of customizable data mining we want to develop. It is important to stress that such a toolset could not only bring the research progression forward, but it could potentially double as an actual modeling asset, usable in future OPP-component library building.