NTNU – Trondheim
Norwegian University of
Science and Technology

Ultra-low voltage embedded processor system for Internet-of-Things microcontrollers

Danton Canut Benemann

Embedded Computing Systems
Submission date: July 2014
Supervisor: Snorre Aunet, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Title:	Ultra-Low Voltage Embedded Processor System for the Internet of Things Microcontrolers
Student:	Danton Canut Benemann

Problem description:

It is predicted that billions of embedded devices will soon be connected to the internet. End nodes will typically contain a microcontroller to collect, process, and interpret sensory input data. Many of these will run from small batteries and may need to harvest energy from the environment to achieve an acceptable battery lifetime. The microcontroller typically includes a CPU, memories, buses, and peripherals. The student should design the embedded processor system within the microcontroller.

The system should be able to run from a wide supply voltage range. The system should be implemented in an low- voltage standard cell library using state-of-the-art design techniques.

The system should support voltage scaling from the nominal voltage down to the subthreshold voltage domain, allowing a wide tradeoff between performance and power consumption. The student should analyze the system properties and compare them to an identical system implemented using traditional libraries and design techniques.

Responsible professor:	Snorre Aunet, NTNU-IET
Supervisor:	Frode Pedersen, ATMEL

Abstract

As devices get an ever increasing foothold on the internet and the Internet of Things becomes the usual landscape in the mass consumer electronic products is necessary to bring embedded processors capable of bringing performance to the micro controllers of such devices. compares Is the intention of this project to compare under different scenarios suitable for the applications two embedded processors form Atmel's one AVR 8-bit and an AVR32 UC3 to identify which is the one that has the lowest power consumption, lowest energy consumption. It has been shown that the AVR has the less power consumption(about half) but in terms of energy the AVR32 is less taxing. A test bench was implemented including SPI communication modules and memories,This is due to the long execution times of the AVR which are for 1.5 to 10 longer than the AVR32. These conclusions scale when implementing on lower feature size (this project uses UMC's 130nm typical process library and UMC's 65 typical process library). Also the conclusion stand when restricting the standard library such that an "Ultra low voltage" friendly net list is produced and tested yielding similar results.

Preface

This Master Thesis was carried out as a part of the study program European Masters in Embedded Computing Systems at Department of Electronics and Telecommunications of the Norwegian University of Science and Technology under the supervision of Prof. Snorre Aunet. The work was done at the Atmel office in Trondheim under the supervision of Frode Pedersen during period between January and July 2014.

Contents

List of Figures vii

List of Tables ix

List of Acronyms xi

1 Introduction 1
- 1.1 Historical Perspective . . . 2
  - 1.1.1 Internet of Things . . . 2
  - 1.1.2 Related Work . . . 2
- 1.2 Asigment Interpretation . . . 3
- 1.3 Contributions . . . 3
- 1.4 Report Organization . . . 4

2 Background 5
- 2.1 Power . . . 5
  - 2.1.1 Dynamic Power . . . 5
  - 2.1.2 Static Power . . . 6
  - 2.1.3 Dynamic Voltage Frequency Scaling (DVFS) and Voltage Islands 7
  - 2.1.4 Near- Threshold Voltage (NTC) and Sub-Threshold Voltage (STC) . . . 7
- 2.2 Heterogenous Computing . . . 8
  - 2.2.1 8/16bit cores vs 32-bit cores . . . 9

3 Implementation 11
- 3.1 Setup and Work Flow . . . 11
  - 3.1.1 Hardware . . . 12
  - 3.1.2 Software . . . 15
- 3.2 Synthesis . . . 17
  - 3.2.1 Synthesis script . . . 17
  - 3.2.2 Libraries . . . 18
- 3.3 Power and Energy Analysis . . . 19
  - 3.3.1 Power . . . 19

3.3.2 Energy . . . 19

4 Results **21**
4.1 Synthesis AVR and AVR32 . . . 21
4.1.1 Library variation . . . 21
4.1.2 Area Comparison . . . 22
4.1.3 Timing Achieved . . . 22
4.2 Power and Energy comparison between the AVR and AVR32 . . . 23
4.2.1 8-bit Math Test . . . 24
4.2.2 8-bit Switch Test . . . 27
4.2.3 16-bit Math Test . . . 30
4.2.4 16-bit Switch Test . . . 33
4.2.5 32-bit Math Test . . . 36
4.2.6 RAM Write Test . . . 39
4.2.7 Serial Peripherial Interface (SPI) Read and Wirte Test . . . 42
4.2.8 Tranfer Control Protocol / Internet Protocol Checksum Computation (TCP/IP Checksum Computation) Test . . . 45
4.2.9 Summary . . . 48

5 Discussion **49**
5.1 Synthesis . . . 49
5.1.1 Area . . . 49
5.1.2 Timing . . . 49
5.2 Power and Energy . . . 50
5.2.1 AVR vs AVR32 . . . 50
5.2.2 Core vs System . . . 51
5.2.3 NTV and STV . . . 51
5.3 Future Work . . . 51
5.3.1 Test Cases and Peripherals . . . 51
5.3.2 Real STV, NTV and More Libraries . . . 52
5.3.3 Tools and Testing Cores at the Same Time . . . 52
5.3.4 Cores . . . 52
5.4 Conclusions . . . 52

References **53**

List of Figures

2.1 NMOS.[24] . . . 6
2.2 Energy per Operation and Delay vs Voltage.[11] . . . 8
2.3 big.LITTLE system.[13] . . . 9
2.4 8-bit Central Procesing Unit (CPU) Comparison[16] . . . 9
2.5 Atmel's Product Range [6] . . . 10

3.1 AVR core (AVR) setup . . . 13
3.2 AVR32 UC core (AVR32) setup . . . 14

4.1 Area footprint . . . 23
4.2 Power Graphs for the 8-bit Math Test . . . 24
4.3 Energy Graphs for the 8-bit Math Test . . . 24
4.4 Power Density Graphs for the 8-bit Math Test . . . 24
4.5 CPU % Power Graphs for the 8-bit Math Test . . . 24
4.6 Power Graphs for the 8-bit Switch Test . . . 27
4.7 Energy Graphs for the 8-bit Switch Test . . . 27
4.8 Power Density Graphs for the 8-bit Switch Test . . . 27
4.9 CPU % Power Graphs for the 8-bit Switch Test . . . 27
4.10 Power Graphs for the 16-bit Math Test . . . 30
4.11 Energy Graphs for the 16-bit Math Test . . . 30
4.12 Power Density Graphs for the 16-bit Math Test . . . 30
4.13 CPU % Power Graphs for the 16-bit Math Test . . . 30
4.14 Power Graphs for the 16-bit Switch Test . . . 33
4.15 Energy Graphs for the 16-bit Switch Test . . . 33
4.16 Power Density Graphs for the 16-bit Switch Test . . . 33
4.17 CPU % Power Graphs for the 16-bit Switch Test . . . 33
4.18 Power Graphs for the 32-bit Math Test . . . 36
4.19 Energy Graphs for the 32-bit Math Test . . . 36
4.20 Power Density Graphs for the 32-bit Math Test . . . 36
4.21 CPU % Power Graphs for the 32-bit Math Test . . . 36
4.22 Power Graphs for the RAM Test . . . 39
4.23 Energy Graphs for the RAM Test . . . 39

4.24 Power Density Graphs for the RAM Test 39
4.25 CPU % Power Graphs for the RAM Test 39
4.26 Power Graphs for the SPI Test 42
4.27 Energy Graphs for the SPI Test 42
4.28 Power Density Graphs for the SPI Test 42
4.29 CPU % Power Graphs for the SPI Test 42
4.30 Power Graphs for the TCP/IP Checksum Test 45
4.31 Energy Graphs for the TCP/IP Checksum Test 45
4.32 Power Density Graphs for the TCP/IP Checksum Test 45
4.33 CPU % Power Graphs for the TCP/IP Checksum Test 45

List of Tables

3.1 Address Map 15

4.1 Area vs targeted time 22
4.2 Timing Achieved for each net-list 23
4.3 Power Comparisons 8-bit Math Test 25
4.4 Energy Comparisons 8-bit Math Test 25
4.5 8-bit Math Energy Estimation for Near-Threshold Voltage (NTV) and Sub-Threshold Voltage (STV) 26
4.6 Power Comparisons 8-bit Switch Test 28
4.7 Energy Comparisons 8-bit Switch Test 28
4.8 8-bit Switch Energy Estimation for NTV and STV 29
4.9 Power Comparisons 16-bit Math Test 31
4.10 Energy Comparisons 16-bit Math Test 31
4.11 16-bit Math Energy Estimation for NTV and STV 32
4.12 Power Comparisons 16-bit Switch Test 34
4.13 Energy Comparisons 16-bit Switch Test 34
4.14 16-bit Switch Energy Estimation for NTV and STV 35
4.15 Power Comparisons 32-bit Math Test 37
4.16 Energy Comparisons 32-bit Math Test 37
4.17 32-bit Math Energy Estimation for NTV and STV 38
4.18 Power Comparisons RAM Test 40
4.19 Energy Comparisons RAM Test 40
4.20 RAM Energy Estimation for NTV and STV 41
4.21 Power Comparisons SPI Test 43
4.22 Energy Comparisons SPI Test 43
4.23 SPI Energy Estimation for NTV and STV 44
4.24 Power Comparisons TCP/IP Checksum Test 46
4.25 Energy Comparisons TCP/IP Checksum Test 46
4.26 TCP/IP Checksum Energy Estimation for NTV and STV 47

List of Acronyms

V_{dd} suplply voltage.

V_{th} threshold voltage.

AHB Advance High Performance Bus.

AMBA Advance Microcontroller Bus Architecture.

APB Advance Peripherial Bus.

ARM Advance Risc Machine.

AVR AVR core.

AVR bus AVR Bus System.

AVR32 AVR32 UC core.

CMOS Complematry Metal Oxide Semiconductor.

CPU Central Procesing Unit.

DSP Digital Signal Processor.

DVE Discovery Visualization Enviroment.

DVFS Dynamic Voltage Frequency Scaling.

GPIO General Purpose Input Output.

HDL Hardware Description Language.

I/O Input/ Output.

IoT Internet of Things.

nMOS n-channel Metal Oxide Semiconductor.

NTNU Norwegian University of Science and Technology.

NTV Near-Threshold Voltage.

pMOS p-channel Metal Oxide Semiconductor.

RISC Reduced Instruction Set Computer.

RTL Register Transfer Level.

SAIF file Switching Activity Interchange File.

SoC System on Chip.

SPI Serial Peripherial Interface.

STV Sub-Threshold Voltage.

TCP Transfer Control Protocol.

TCP/IP Checksum Computation Tranfer Control Protocol / Internet Protocol Checksum Computation.

VCD file IEEE standard waveform database dumpfile.

VCS Verilog Compiler Simulator.

VPD file Synopsys waveform database dumpfile.

Chapter 1
Introduction

In today's world the number of devices featuring features enabled by embedded systems is increasing. At the core of these devices there is a System on Chip (SoC) consisting of at least a CPU a General Purpose Input Output (GPIO), a memory and a bus connecting all. From the user /environment perspective the goal of these SoC is to react to stimulus on the GPIO. In a well implemented system the reaction to the stimuli is a appropriate according to a program stored in the memory ran by the CPU. The human machine interfaces for the majority of this systems are very limited or non existent. But these machines are the cornerstone of modern consumer products which features have become standard of living in modern industrialized societies.

At the same time there is a huge pressure towards have higher integration between devices by means of sharing data, collaborating to solve tasks, accessing remote peripherals, producing data, etc. All of that will enable the implementation of interesting and useful features such as load balancing in an electric grid by timing when and where electric consumers can get online and how much power it can be drawn form there. For this to happen SoC besides the CPU, memory GPIO and bus it must contain at least one communication module that enables it to communicate with other devices. And the vast majority of this embedded systems will have limited access to electrical power.

Since the set of tasks that is expected to be performed by the embedded systems varies a lot depending on the application one might be tempted to think that different CPUs will perform better than others performing certain tasks. Parallel to that it is known that the amount of power used by any electronic device is proportional to the squared of the voltage. The juxtaposition of the previous two statements mean that in order to implement an efficient device whose energy consumption is as low as possible -so it can be operated with a constrained power budget like a small battery or by harvesting energy- It is necessary to study different CPUs performing the same tasks. It is also necessary to watch the effect that the use of different

implementations of different on the energy consumption. The goal of this thesis to observe the energy consumption in an 8-bit implementation of Atmel's popular AVR core and in a 32-bit AVR32 and to quantify the impact different process technology and reduced library more suitable to a ultra low voltage library.

1.1 Historical Perspective

1.1.1 Internet of Things

In 1999 Kevin Ashton was the first person to use the term Internet of Things (IoT) [2] during a presentation for Procter & Gamble identifying the potential of integrating the RFID technology in the supply chain to gather data and the internet. He points out that most of the information available on the internet has been either been input by a human or triggered by a human action. The accuracy, consistency and flow of this data then is dependent of human accuracy, consistency and latency. Thanks to RFID technology and similar technologies data can be automatically be uploaded to the internet with the accuracy, consistency and latency of machines. Is also Ashton's view that this automatic data gathering capabilities and easy access thru the internet will empower machines to reduce waste, increase efficiency and alert when maintenance task should be performed.

Meanwhile in 2004 an article published on Scientific American [21] made the analogy of the segmentation of different communication protocols for devices to comunicate such as ZigBee, Bluetooth, etc. to the Arpanet and other Internet predecessors in the 1960's and makes a very good case why devices should use the same internet standards to exchange information. Among the main advantages identified by the article are packet switched networks, unique addressing for each nodes

1.1.2 Related Work

This work builds on the idea to use the AVR as a coprocessor for the AVR32 exposed in the master thesis of M Sc. Yahsir Mahmood[20]. It is also used a modified version of his test bench for benchmarking the AVR32. On his thesis M. Sc Mahmood goes thru the process of designing an energy efficient AVR Bus System (AVR bus) to Advance Peripherial Bus (APB) needed to access the peripherals with the AVR and the AVR32[20]. In this work however instead of using the AVR bus to APB bridge an AVR bus to Advance High Performance Bus (AHB) bridge is used as it enables the AVR to access the same high performance RAM as the AVR32 and puts the AVR on the same heretical level as AVR32.

1.2 Asigment Interpretation

The following tasks were proposed and executed to address the problem:

Task 1: Set up a test bench that connects RAM, SPI communication modules and other periferials via an AMBA AHB lite bus with a CPU.

Task 2: Test access to memory and peripherals connected in task 1 using an AVR and an AVR32 and confirm it function correctly.

Task 3: Choose a set of test programs to benchmark that perform tasks of a simple but representative IoT application.

Task 4: Synthesize the set-ups used for task 2 in different feature sizes and with different constraints.

Task 5: Perform power analysis on the results of task 4 running the test programs established on task 3.

Task 6: Compute the energy used to perform each test program under each net-lists.

Upon completion of these tasks data the following objectives are achieved:

Objective 1: Identify the effect on power and energy consumption that constraining the standard cells that the synthesis tool can use under different scenarios.

Objective 2: Identify which scenarios favor the architecture of the AVR32 and which favor the aAVR.

Objective 3: Test if the conclusions from objective 2 stands if feature size is changed.

Objective 4: Extrapolate the energy and power figures if implemented on an ultra low voltage library.

1.3 Contributions

A test bench that accepts any core that interfaces thru AHB bus. Eight different test relevant to the IoT scenario to better understand the power, energy consumption of the AVR and AVR32. A comparison consisting of 640 experiments where libraries and timing contains and feature size core and application were varied to monitor the power and energy consumption and an estimate of what to expect when moving to the STV and NTV domains.

1.4 Report Organization

This report is organized as:

Chapter 1: Introduction Introduces the master thesis, provides some historical perspective, states how the problem was interpreted, how it was approached, summarizes the contributions of the thesis and shows how the report is organized.

Chapter 2: Background Provides the theoretical background on the components of power on a Complematry Metal Oxide Semiconductor (CMOS) circuit, examines some techniques to save power while paying performance penalty such as Dynamic Voltage Frequency Scaling (DVFS) and voltage islands and explains the potentials saving that can be achived by reducing the voltage near and below the threshold region. Also comments on the advantage of heterogenous computing and shows two commercial examples.

Chapter 3: Implementation Describe the work flow that was followed, the components that were used, how were used, the tests that were performed and how they were performed the tools that were used for testing.

Chapter 4: Results Shows the result obtained by the methodology proposed. Address each test and extracts and directs attention to interesting facts derived from the results .

Chapter 5: Discussion Analyzes the results and provides some explanations .

Chapter 2
Background

2.1 Power

There are two sources of power dissipation by a CMOS circuit [22]

$$P_{total} = P_{dynamic} + P_{static} \tag{2.1}$$

2.1.1 Dynamic Power

Dynamic power is consumed when changing the logic level the inputs of a gate, this causes the n-channel Metal Oxide Semiconductor (nMOS) and p-channel Metal Oxide Semiconductor (pMOS) transistors to switch. From figure it is appreciated that any switching activity will cause that the pMOS and the nMOS transistors are in the *conducting* state at the same time causing a *short circuit current* to flow between suplply voltage (V_{dd}) and *GND*. At the same time the effective load capacitance is discharged or charged causing further power consumption. [22]

$$P_{dynamic} = P_{switching} + P_{short\ circuit} \tag{2.2}$$

It is possible to express:

$$P_{short\ circuit} = I_{short\ circuit} V_{DD} \tag{2.3}$$

Where $P_{short\ circuit}$ is the average short-circuit power, $I_{short\ circuit}$ is the average short-circuit current.

For quantifying the power dissipated by the switching activity we need to know two quantities the effective capacitance driven by the gate and how often this capacity (switching frequency) is charge/discharge.The effective capacitance is dependent of the gate, interconnect and the load. The switching frequency is dependent of the logic and clock frequency. After some mathematics and analysis of the circuit is possible to express the average swtiching power as: [22]

$$P_{swtiching} = \alpha C V_{DD}^2 f \tag{2.4}$$

Where $P_{swtiching}$ is the average swtiching power, α is the *activity factor* that expresses in terms of the clock frequency the charging/ discharging, C is the *effective capacitance* and f is the clock frequency.

2.1.2 Static Power

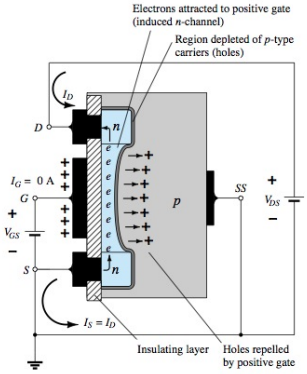


Figure 2.1: NMOS.[24]

Static power for CMOS has three main contributors. Subthreshold leakage, gate leakage drain and source diffusion leakage.[22]

$$P_{static} = (I_{sub} + I_{gate} + I_{junct})V_{DD} \tag{2.5}$$

Figure 2.1 shows the junction of semiconductor materials that form an pMOS transistor. Thru these junctions and thanks to minority carriers there is a leakage current. The gate leakage can be reduced by substituting the silicon dioxide with high dielectric materials between the gate and the semiconductor substrate.[12] Leakage of the junctions can be reduced by doping [1]and voltage on the body[8].

When operating on the subthreshold region the MOS transistor then can be modeled as bipolar transistor as shown by Figure bla the and equation 2.6 describes the behavior of the current flowing from drain to source.[22]

$$I_{sub} = I_{ds0} e^{\frac{V_{gs} - V_{t0} + \eta V_{ds} - \kappa_\gamma V_{sb}}{n v_T}} \left(1 - e^{-\frac{V_{ds}}{v_T}}\right) \tag{2.6}$$

Where I_{ds0} is the current at threshold it is dependent of process and geometry, V_{gs} is the voltage between the gate and source, V_{t0} threshold voltage when $V_{source} = V_{body}$, η DIBL coefficient, V_{ds} voltage between drain and source, κ_γ body effect (geometry, dielectric, surface potential, etc), V_{sb} Voltage between source and body, v_T Termal voltage. [22]

From equations 2.5 and 2.3 have a linear dependency to V_{DD}, but is worth noticing that I_{sub} is exponentialy dependant of V_{ds} at the same time equation 2.4 shows a quadratic dependency to V_{DD}. Then it is shown that lowering the operating voltage is a way to save power. Nevertheless reducing the voltage can degrade the performance as it reduces de current that goes from drain to source and with that the time that is required to drive the loads increasing delays.

2.1.3 Dynamic Voltage Frequency Scaling (DVFS) and Voltage Islands

DVFS scaling is a technique used to save power. It reduces de voltage and frequency when the work load is low (meaning the timing constrains are more relaxed) and increases the voltage and frequency when the work load is high (meaning the timing contains are tighter). For achieving an energy efficient implementation it is required that voltage/ frequency regulator is an efficient one, the granularity of the voltage and frequency to choose from allow to adapt at any processor workload. The main goal is to consume less energy, this means that the power savings for a task must be greater than the time increase for the same task.[7]

It is possible to have different voltages on a SoC this technique is known as voltage islands. Each island operates on a different voltage level .This technique allows blocks to be grouped together on an island where the voltage presents the right compromise between the power savings and performance degradation. It does not require the regulation overhead that DVFS and it is not aware of the workload situation. [14]

2.1.4 Near- Threshold Voltage (NTC) and Sub-Threshold Voltage (STC)

As stated in the previously, reducing the voltage dramatically reduce the power consumed. Figure 2.2 shows the relationship between the energy spent for operation, the delay and the supply voltage. More energy is saved from the transition from super-thresholdvoltage to NTV than form NTV to STV this is due to the increase in leakage that is observed when $V_{dd} <$ threshold voltage (V_{th}). The same figure also shows an exponential growth on the delay in the STV region. It is of particular interest the existence of a minimum on the energy per operation curve that indicates there is a voltage in the sub-threshold region that is optimum form the energy perspective but comes at the cost of increased delay. [11]

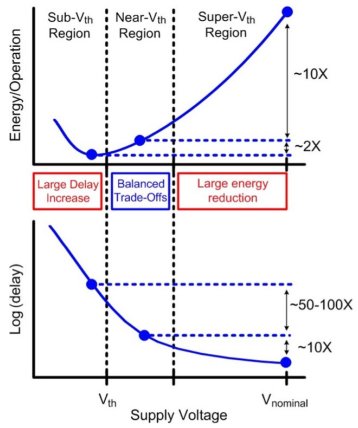


Figure 2.2: Energy per Operation and Delay vs Voltage.[11]

2.2 Heterogenous Computing

Kummar et al. have shown that higher energy efficiency can be achieved on a SoC by using heterogenous cores.[17] By definition an heterogenous core system is a multi-core system made up by different cores. There are difference in terms of energy efficiency, performance, power consumption and capabilities between different cores. There are energy efficient cores that are suitable for simple operations and high performance cores capable of performing complex task but the power consumption is high. Having both on the same SoC allow to have the best of both worlds.

Industry have taken notice of this and there are examples of heterogenous muli-cores that feature an energy efficient core and a high performance core being deployed such as:

Advance Risc Machine (ARM) exploits the energy efficiency of the Cortex- A7 core (LITTLE) and the high performance of the Cortex- A15 (big) in its *big.LITTLE* architecture shown on Figure 2.3. ARM is quick to notice that despite labeling the Cortex-A7 as *LITTLE* it features micro-architecture advances that culminates on performance than implementations of the Cortex-A8 at a fraction of the power.[13]

NVIDIA uses a power optimized Cortex A9 core(companion core) and four performance optimized Cortex A9 cores (main cores) in its *Tegra 3* formerly known as *Kal-El*. The companion core is limited to 500MHz [23]

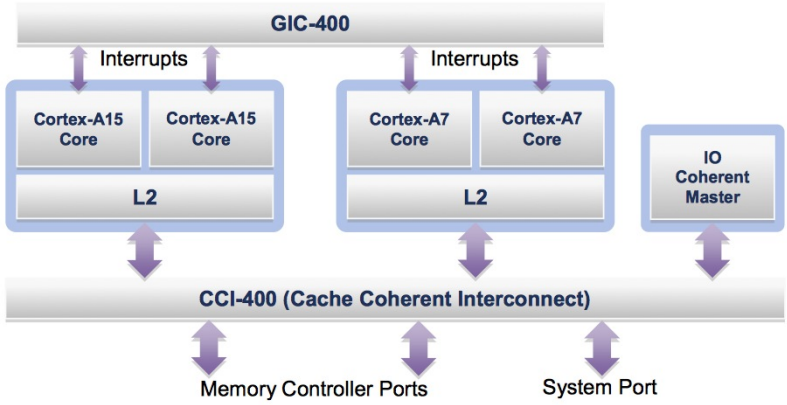


Figure 2.3: big.LITTLE system.[13]

2.2.1 8/16bit cores vs 32-bit cores

Since the smbedded systems environment is diverse, system performance is difficult to define. Some applications require realtime guarantees, others require high precision and/or fast measurement peripherals, others have very limited energy/power contains and finally there are the ones that require lots of computing power.[16] Figure 2.4a shows the benchmark obtained after running the code on Figure 4.2b. It is shown that the AVR is a good candidate to be considered for be a *LITTLE* like co- processor. For an heterogenous system. 8/16 bit processors have the problem that they not support

CPU Architectures	Code size (bytes)	Execution time (cycles)
8051	112	9384
PIC16	87	2492
AVR	46	335

(a) Code Size and Execution

```
int max(int *array)
{
  char a;
  int maximum=-32768;

  for (a=0;a<16;a++)
    if (array[a]>maximum)
      maximum=array[a];
  return (maximum);
}
```

(b) Sample Code

Figure 2.4: 8-bit CPU Comparison[16]

large memory space since the mayority of them feature only 16.bit address bus. This an other limitations have open the door for 32-bit cores that might be not the right solution due its increase complexity, Nevertheless there are application areas that require some of the extra performance offered by the 32-bit alternative for a small part

of the application providing further motivation for an heterogenous architecture.[16] Figure 2.5 Shows Atmel's portfolio for 8-bit and 32-bit micro controllers.

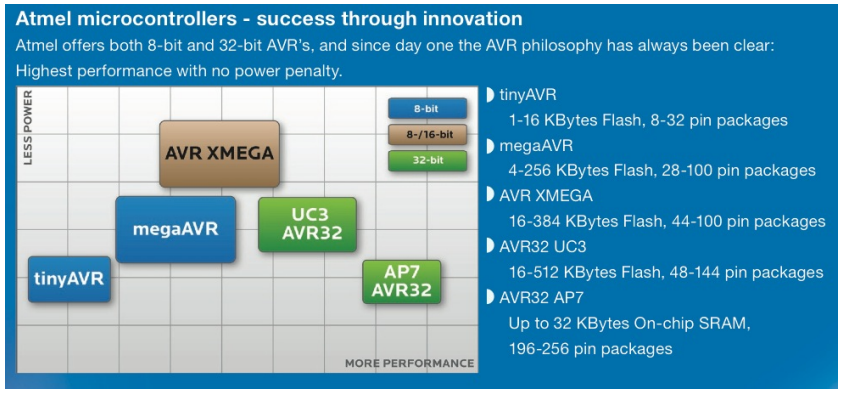


Figure 2.5: Atmel's Product Range [6]

Chapter 3
Implementation

This chapter is divided in three sections. Setup and Work Flow covers the hardware and test cases used for this project. Synthesis explains how the synthesis was performed and declares the libraries used to produce the net-lists on which the power analysis was performed. And the power analysis section go thru the generation of the reports and data gathering and manipulation. Data that is presented in Chapter 4

3.1 Setup and Work Flow

This section is divided into two main parts. The first part describes the hardware used and the second describes the software used. It is very important to note that there is no silicon used for this project.

The basic work flow is as follows:

Step 1: The hardware is defined at the RTL label and a test program is prepared.

Step 2: Synopsys' Verilog Compiler Simulator (VCS) to produce a Synopsys waveform database dumpfile (VPD file).

Step 3: Verify the correct functionality using Discovery Visualization Enviroment (DVE).

Step 4: Test case is prepared on C or asm and compile them with AVR-toolchain or AVR32-toolchain (depending on the targeted processor) to produce the executable.

Step 5: Use executable, VCS, and DVE to check function, start and end times of the program or interesting subroutine.

Step 6: Use Synopsys' design complier and synthesis script to produce the appropriate net-list and save the it for future usage.

Step 7: Transform the VPD file into a IEEE standard waveform database dumpfile (VCD file) and this into a Switching Activity Interchange File (SAIF file) for this last transformation the tool requires the start and end time of interest.

Step 8: Use Synopsys' design compiler and power analysis script with the net-list generated on step 6 and the SAIF file on the step 7

Step 9: Extract and analyze the data outputted on the power report generated on step 8.

3.1.1 Hardware

In order to be able to compare the energy consumption of the AVR32 and the AVR they must be tested under the same circumstances. Figure 3.2 and 3.1 show the test environments for both processors. As it is shown the test environments are as similar as it can be. The only mayor difference is that the AVR setup includes a bridge AVR bus to AHB. This bridge is not necessary for the AVR32 since it has native interface for AHB.

3.1.1.1 AVR32 UC core

The AVR32 is a Super Harvard architecture 32-bit processor that features a compact Reduced Instruction Set Computer (RISC) instruction set including Digital Signal Processor (DSP) instructions fully orthogonal. This processor has two AHB interfaces one for the program memory and other for data and peripheral communication, RAM interface, 15 32-bit wide general purpose registers 32-bit wide stack pointer, program counter and link register. It is a 3- stage pipeline processor allowing one instruction per clock cycle for most instructions.[5]

3.1.1.2 AVR core

The AVR is a Harvard architecture 8-bit RISC processor. It has 32 registers 8-bits wide, The last six is are concatenated in pairs to form a 16-bit wide pointers to address the data space. Since it is a Harvard architecture the program memory and the data memory are not only physically separated but they have separate address spaces, separate data and address lines and the data memory is accessed by using the same instructions as the Input/ Output (I/O). [4]

The AVR data bus is synchronous based on master/ slave scene and has 16-bit for addresses, 8-bit for data form the master, 8-bits for data from the slave, write, read, wait, single bus cycle and burst signals. This bus allow single access operation, this means that on the same clock cycle the master puts the address on the address lines and either send data or receives it and is ready for another operation on the next clock cycle if there is no wait signal form the slave.[3]

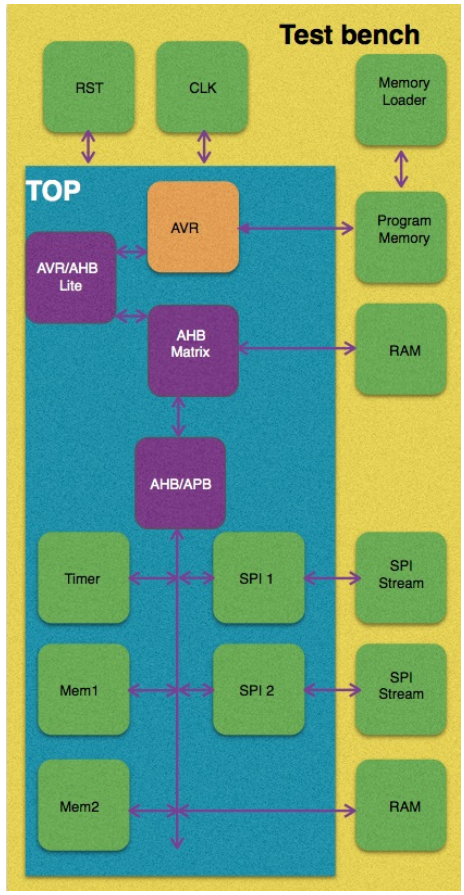


Figure 3.1: AVR setup

3.1.1.3 Advance High Performance Bus

AHB is a multi master bus protocol part of the Advance Microcontroller Bus Architecture (AMBA) family of open protocols for on-chip communications created by ARM. AHB is the high performance bus that supports high clock frequency. It is a pipelined bus that uses the master/ slave scheme which means the master on the first clock cycle put the address and the appropriate control signals and the next clock cycle it expects a reply form the slave (transfer complete, error, wait state or split transaction) and puts the next address on the bus. [18]

For the scope of this project it is used the single master AHB lite standard which is a simpler implementation[19] and the bridge implemented is constrained to simple

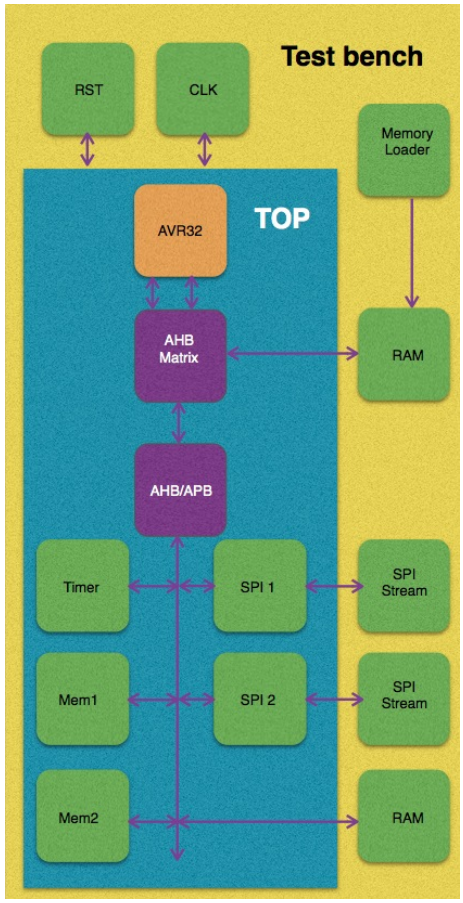


Figure 3.2: AVR32 setup

write [1] and simple read that allow the slaves to add wait states.

3.1.1.4 Top and Test bench

Figure 3.2 and Figure 3.1 shows what is instantiated at the top level and how it interfaces with the test bench. The top-level include the processor core (AVR or AVR32) AHB, APB busses -in the case of the AVR the AHB to AVR bus- two SPI modules a timer, two memory modules. The test bench has the clock generator, RAMs, program memory and the program memory loader that takes the .hex file and loads the memory with it.

[1] Due the mismatch between the 32-bit wide data bus from the AHB and the 8-bit wide data bus from the AVR bus all write operations are a read/ modify operations

Peripherial	Base Address
AHB RAM	0x00000000
APB RAM	0xFFFF0000
Timer	0xFFFF2000
SPI 1	0xFFFF3000
SPI 2	0xFFFF4000
Mem 1	0xFFFF0000
Mem 2	0xFFFF0000

Table 3.1: Address Map

Table 3.1 shows the peripherals with it is corresponding Base Address.

3.1.2 Software

This section explains what the each test do. All this were written in C (except for the case of the SPI for the AVR which was written in assembly) and complied by the AVR toolchain and AVR32 toolchain to produce the executable code needed by the processors.

All the test were proposed thinking on an IoT scenario were a micro controller gets data from a sensor via a communication peripheral (SPI was chosen due its simplicity and huge popularity), the communication is bidirectional -the data streams are simulated by the test bench- performs some arithmetic manipulation (addition correcting systematic error and multiplication for sensitivity error correction), testing this value against a parameter and use Transfer Control Protocol (TCP) to connect to the internet. Due to board range of applications for the arithmetic manipulation three tests were proposed -8, 16, and 32 bit- and due to the unknown amount of cases to be handled two tests were proposed -8 and 16 bit-. The TCP is a complex protocol and with the support given by media access hardware undefined it is difficult to specify what to test. The computing the check sum of a TCP packet was chosen due to its importance when determining data integrity and assuming simple hardware that implements complete functionality for layers 1 to 3 of the OSI model.

3.1.2.1 8-bit Math Test

This test in its main defines two 8-bit operands and calls a function that adds and a function that multiplies. It takes 4340ns for the AVR and 1220ns for the AVR32 to finish.

3.1.2.2 8-bit Switch Test

This test in its main defines a 8-bit operand and calls a function that selects the case and returns a 8-bit value to the main. It takes 1700ns for the AVR and 980ns for the AVR32 to finish.

3.1.2.3 16-bit Math Test

This test in its main defines two 16-bit operands and calls a function that adds and a function that multiplies. It takes 5680ns for the AVR and 1340ns for the AVR32 to finish.

3.1.2.4 16-bit Switch Test

This test in its main defines a 16-bit operand and calls a function that selects the case and returns a 16-bit value to the main. It takes 2120ns for the AVR and 980ns for the AVR32 to finish.

3.1.2.5 32-bit Math Test

This test in its main defines two 32-bit operands and calls a function that adds and a function that multiplies. It takes 10720ns for the AVR and 1160ns for the AVR32 to finish.

3.1.2.6 RAM Write Test

This test in its main performs 100 RAM writes. It takes 30340ns for the AVR and 6640ns for the AVR32 to finish.

3.1.2.7 SPI Read and Wirte Test

This test its main sets up SPI 1, SPI 2, then performs 10 times a SPI read on SPI 1, a multiplication, 8-bit shift and a write on SPI 2. The program for the AVR first configures the bridge to access the I/O space and then proceeds to do the same as the program for the AVR32 and after that it configures back the bridge to address the memory space. It takes 18820ns for the AVR and 8560ns for the AVR32 to finish.

3.1.2.8 TCP/IP Checksum Computation Test

This test in its main computes the check sum field of a 120 byte TCP frame. which is a set of 16-bit sum, shift and byte swap operations. It takes 37220ns for the AVR and 7900ns for the AVR32 to finish.

3.2 Synthesis

The goal of the synthesis is to translate the Register Transfer Level (RTL) code to a net-list of interconnected logic gates chosen from a library.

The synthesis is done by the Synopsys' design compiler following the instructions and contains stated by the synthesis scripts.

Performing the synthesis is paramount for this project. Changes done on the constrains and library selection produced the variation in results that will be presented on Chapter 4 and discussed on Chapter 5. To produce a net-list a library is chosen, a clock period is targeted and a maximum area is defined. For all cases the maximum area set to zero knowing it will be impossible to the tool to achieve this but will force the tool to get the smallest possible, the tool is also instructed to do its best effort. Per each library used ten clock periods were targeted, form 1ns to 10ns with a step of one nano second. Aditionaly the tool is intructed to keep the hierarchy -this is don e so the power analisys is heriarchial as well-

3.2.1 Synthesis script

The script commands the tool to do:

Step 1: Analyze.

Step 2: Elaborate.

Step 3: Link.

Step 4: Load the library with the standard cells.

Step 5: Load the constrains (area, timing, wire model).

Step 6: Compile the design (clock gating is performed at this stage).

Step 7: Write net-lists.

The set of steps defined before will produce a net list that would be used on power and energy analysis described later this chapter. Ideally for this project a STV or NTV library should be used for this project but no such library was available so a super threshold voltage library was restricted to simulate the restriction in terms of gates that are present in a subthreshold voltage libraries.

The synthesis tool produces the following reports:

Area: Reports the area taken bay the combinational logic, non combinational and sequential and the hierarchical distribution.

Clock Gating: Shows the elements that are clock gated with its inputs, outputs and related registers. Presents the elements that were not able to be clock gated and the violation that will ocur if gated. It also displays a summary with the number of clock gating elements, the number of registers gated, the number of registers ungated.

Resources: Reports on the resources shared by the modules, the implementations and the multiplexors added.

Power: Presents a power consumption estimate.

Timing: Shows critical paths, different points on the path and the time it takes from point to point. Gives a summary on the required data time and the arrival of that data and computes the slack.

References: Reports the cells that were used, library of origin, area per cell number of cells used and the total area that type of cell takes on the design

All Violators: Reports on the modules that violate the timing and the area constrictions and by how much it is missed.

3.2.2 Libraries

Two base libraries were used to produce the net-list used in this thesis. UMC's 130nm FSC0H_D generic core and UMC's 65nm FSE0K_D generic core. To see the effect of feature size on energy consumption. For the understanding the effects of a 130nm low voltage library two the cells available for the synthesis tool in two different stages. Both of this libraries are super-threshold voltage. There was no sub-threshold voltage nor near-treshold voltage available.

3.2.2.1 UMC′s FSC0H_D Generic Core

No restrictions

This is a high performance 130nm library that provide support arithmetic cells for data-path design. The corner used was typical process 1.2 V at 25^{o}C with a gate density of 250000 gates per mm^2 featuring a drawn gate length of 0.12μm with a power consumption of 6nW/MHz/gate[9].

Restricted (ltd)

For these synthesis the tool was not allowed to use any gate with more inputs than two and the maximum outputs is two as well. This is the most restricted the library can get.

Restricted (ltd2)

For these synthesis the tool was allowed to use the same gates as the (ltd) case plus gates with a fan-in and fan-out of three.

3.2.2.2 UMC′s FSE0K_D Generic Core

This is a low leakage 65nm library that provide support arithmetic cells for data-path design. The corner used was typical process 1.2 V at 25^oC with a gate density of 900000 gates per mm^2 featuring a drawn gate length of $0.06\mu m$ with a power consumption of 1.9nW/MHz/gate[10].

3.3 Power and Energy Analysis

3.3.1 Power

Before doing the power analysis two things have to be ready. A net list produced by the synthesis tool and a SAIF file that contains the switching activity caused by a test program. For the power analysis again Synopsys' design complier is used with a report power script. The power script first it loads the library data base, it is very important to use the same library used during the synthesis. Then it reads the SAIF file and reports the power used by each component of the top level and the total power.

3.3.2 Energy

The energy is obtained by multiplying the power (average power) results obtained by the report of the power analysis and multiply it by the time spent on the task (the time difference between the end time and start time used for generating the SAIF file) both the energy and power results are only valid for the test case being studied on the net list tested, since at its core it was generated with the switching activity the program caused. For purposose of this thesis the effects described on Section 2.1.4will be estimated a black box and the results descriibed shown by Figure 2.2 apply directly system wide. By using the (3.1) the total energy consumption is estimated where Δ energy consumption factor expresed as the ratio of the energy consumed when changing form the super-threshold voltage to the NTV or STV this wil yield to E_{total} being the energy consumed by the system under the NTV or STV conditions.

$$E_{total} = \Delta E_{total\ Super} \tag{3.1}$$

Based on Johnsen preliminary results[15] for the design of the *Full-Custom SubNear-Threshold Cell Library in 130nm CMOS* the values used for the STV case (350mV) $\Delta = 0.072$ and for the NTV case (400mV) $\Delta = 0.094$. Only the result produced by netlist wiht a target clock period of 10ns will be used for the estimation. The net-list 130nmltd will be used for NTV and STV and the 130ltd2 ones for NTV.

Chapter 4
Results

In this section the results of the synthesis process with the different restrictions and the power analysis data will be presented in graph.

4.1 Synthesis AVR and AVR32

Ten different clock periods were targeted to produce ten net lists for each core set-up (AVR or AVR32) and library (UMC's FSC0H_D Generic Core -130nm-, UMC's FSC0H_D Generic Core Restricted -130nmltd-,UMC's FSC0H_D Generic Core Restricted -130nmltd2- or UMC's FSE0K_D Generic Core) a total of 80 net lists-65nm-.

4.1.1 Library variation

4.1.1.1 UMC's FSC0H_D Generic Core Unrestricted (130nm)

This is the base case. Against the results obtained by this set of net lists comparisons will be made since all the available cells are used and power optimization technique clock gating was done successfully.

4.1.1.2 UMC's FSC0H_D Generic Core Restricted (130nmltd)

This case is the best substitute candidate for a subthreshold voltage since all the gates with fan-ins and fan-outs larger than two were restricted.The clock gating cells were restricted so no clock gating was not performed. This case also takes the largest surface on the wafer.

4.1.1.3 UMC's FSC0H_D Generic Core Restricted (130nmltd2)

This case is the best substitute candidate for a STV since all the gates with fan-ins and fan-outs larger than three were restricted.The clock gating cells were restricted

so no clock gating was not performed. This case also takes the largest surface on the wafer.

4.1.1.4 UMC's FSE0K_D Generic Core (65nm)

This case give us information on how power, area and timing scales for the super-threshold operation it is important as a reference point to observe the effect that feature size has on power and energy. This case also takes the smallest surface on the wafer.

4.1.2 Area Comparison

Figure 4.1 shows that for every case and every core restricting the clock period below certain point causes the area to increase -around 4 ns for the AVR32 and around 3 ns AVR. Table 4.1 show some interesting comparisons that help us to watch the effects of the library constriction and core selection.

Target	130nm		130nmltd		130nmltd2		65nm	
$(ns/\mu m^2)$	AVR	AVR32	AVR	AVR32	AVR	AVR32	AVR	AVR32
1	160634.	347241.	316990.	714925.	301690.	675180.	41094.4	99675.8
2	147715.	343153.	313741.	716474.	295957.	677208.	38298.6	92870.7
3	143369.	336020.	296455.	697892.	283271.	655172.	33063.	73243.8
4	130395.	284294.	285102.	636489.	273185.	609820.	30820.8	65494.4
5	125878.	269527.	280718.	625014.	270459.	598123.	30456.6	63887.7
6	124987.	266066.	280163.	619191.	269422.	596939.	30056	62560.3
7	124535.	264314.	279270.	614902.	269146.	587792.	29964.8	61721.6
8	124333.	260168.	279156.	608965.	268594.	587566.	29899.8	61392.3
9	124255.	256329.	279073.	607360.	268558.	582834	29817.9	61009.3
10	124250.	255328	279068.	607254.	268501.	581165.	29815.4	60802.2

Table 4.1: Area vs targeted time

4.1.3 Timing Achieved

Table 4.2 shows that for every case and every core restricting the clock period below certain point causes the synthesis tool to fail to achieve the timing goal -around 4 ns for the AVR32 and around 3 ns AVR and there is minimum clock period achieved -around 3.1 ns for the AVR32 and 2 ns for the AVR using 130nm feature size-. For the 65nm library the minimum clock period is smaller -around 1.7 ns AVR and 2.4 ns for the AVR32-.

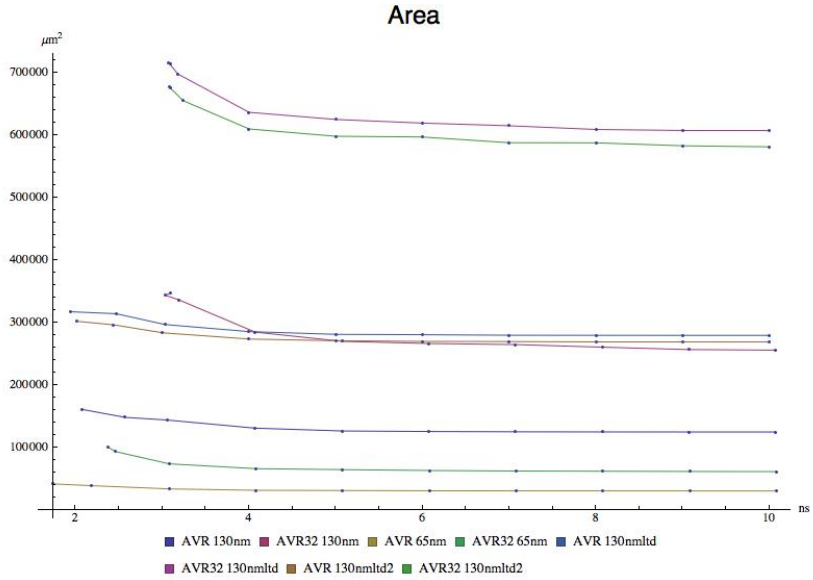


Figure 4.1: Area footprint

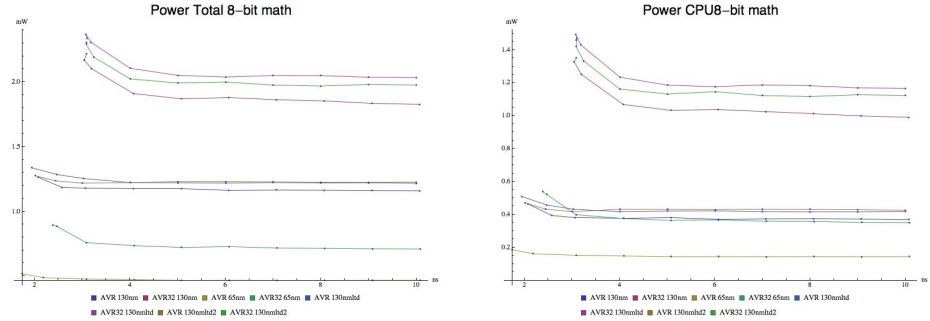
Target	130nm		130nmltd		130nmltd2		65nm	
(ns)	AVR	AVR32	AVR	AVR32	AVR	AVR32	AVR	AVR32
1	2.07	3.09	1.94	3.1	2.01	3.09	1.74	2.38
2	2.57	3.04	2.47	3.07	2.44	3.08	2.18	2.46
3	3.06	3.19	3.03	3.18	3	3.24	3.08	3.08
4	4.07	4.07	4	4	4	4	4.08	4.08
5	5.07	5.07	5	5	5	5	5.08	5.08
6	6.07	6.07	6	6	6	6	6.08	6.08
7	7.07	7.07	7	7	7	7	7.08	7.08
8	8.07	8.07	8	8	8	8	8.08	8.08
9	9.07	9.07	9	9	9	9	9.08	9.08
10	10.07	10.07	10	10	10	10	10.08	10.08

Table 4.2: Timing Achieved for each net-list

4.2 Power and Energy comparison between the AVR and AVR32

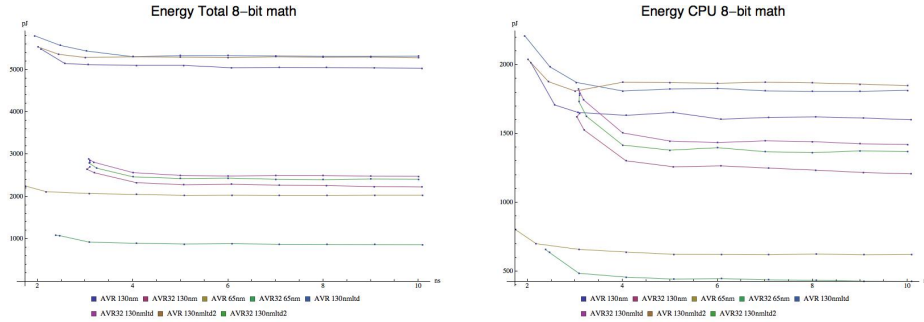
4.2.1 8-bit Math Test

4.2.1.1 Super Threshold Voltage



(a) **(b)**

Figure 4.2: Power Graphs for the 8-bit Math Test



(a) **(b)**

Figure 4.3: Energy Graphs for the 8-bit Math Test

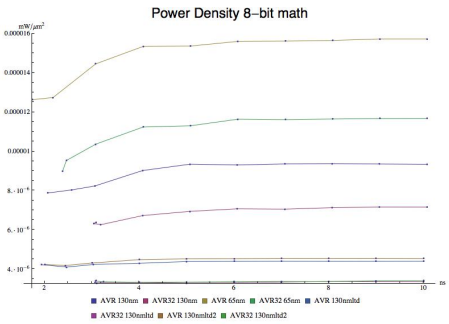


Figure 4.4: Power Density Graphs for the 8-bit Math Test

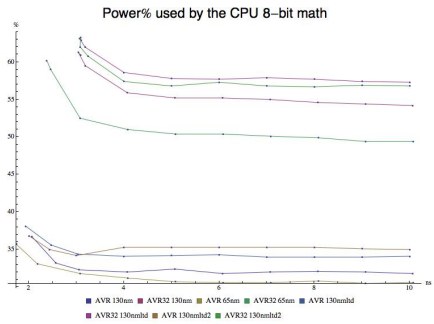


Figure 4.5: CPU % Power Graphs for the 8-bit Math Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	1.83	1.58	1.65	3.38	2.68	2.87
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	1.84	1.66	1.71	3.32	2.79	2.93
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	1.85	1.61	1.69	3.28	2.59	2.8
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	1.81	1.51	1.59	3.23	2.44	2.63
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.05	1.1	1.18	1.09	1.15
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.04	1.06	1.13	1.07	1.1
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.36	0.39	0.4	0.32	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.08	1.04	1.06	1.16	1.1	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.05	1.01	1.04	1.16	1.01	1.13
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.41	0.4	0.4	0.41	0.38	0.39

Table 4.3: Power Comparisons 8-bit Math Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	0.51	0.44	0.46	0.95	0.75	0.81
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	0.52	0.47	0.48	0.93	0.78	0.82
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	0.52	0.45	0.47	0.92	0.73	0.79
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	0.51	0.43	0.45	0.91	0.69	0.74
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.05	1.1	1.18	1.09	1.15
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.04	1.06	1.13	1.07	1.1
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.36	0.39	0.4	0.32	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.08	1.04	1.06	1.16	1.1	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.05	1.01	1.04	1.16	1.01	1.13
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.41	0.4	0.4	0.41	0.38	0.39

Table 4.4: Energy Comparisons 8-bit Math Test

In Figure 4.2a it is observed that the power consumption for the systems using the AVR is very similar for the 130nmltd2 and 130nmltd cases when targeting clock periods larger than 4ns, when targeting periods smaller there is a transition around 3ns and in the vicinity of 2ns the 130ltd2 behaves similar to the 130nm case mainly because the power consumption of the later grows faster than the 130nmltd. Figure 4.2b shows that the AVR 130nmltd2 core consumes more power than its counterparts the but the effect described earlier it is present here as well and around the 3ns it is surpassed by the 130nmltd. The power consumption of the AVR32 65nm is comparable with the power consumption with the implementations of the AVR cores using the 130nm library. Figure 4.3a shows that systems with AVR32 cores consume half the energy than the ones based on the AVR core, however Figure 4.3b shows that if we compare just cores AVR32 cores energy expenditure is only 80% (on average) the energy used by the AVR cores. Figure 4.5Timing restriction around 3ns produces AVR core that the smallest power fraction under the 130nmltd2 case.

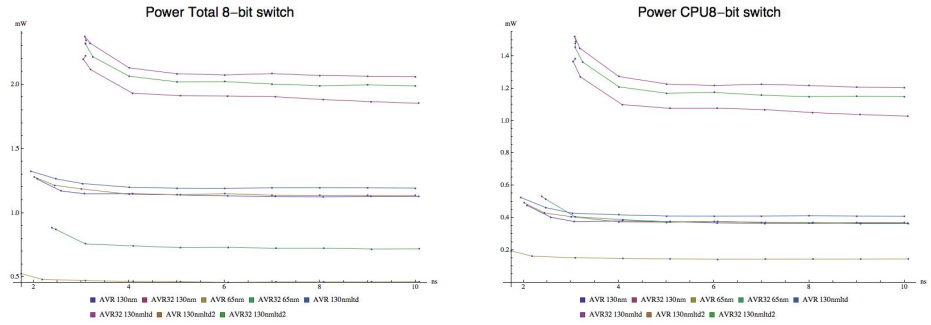
4.2.1.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	2035.46	5034.4	5281.78	5320.84
NTV	—	—	496.487	500.159
STV	—	—	—	383.1
AVR32				
Super TV	866.2	2228.94	2411.94	2481.48
NTV	—	—	226.722	233.259
STV	—	—	—	178.667

Table 4.5: 8-bit Math Energy Estimation for NTV and STV

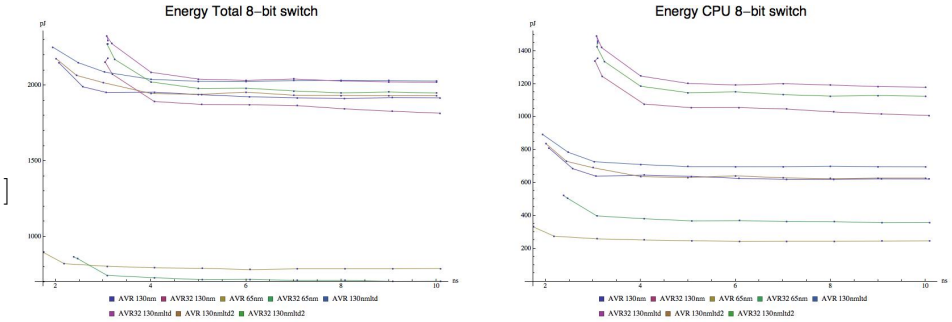
4.2.2 8-bit Switch Test

4.2.2.1 Super Threshold Voltage



(a) **(b)**

Figure 4.6: Power Graphs for the 8-bit Switch Test



(a) **(b)**

Figure 4.7: Energy Graphs for the 8-bit Switch Test

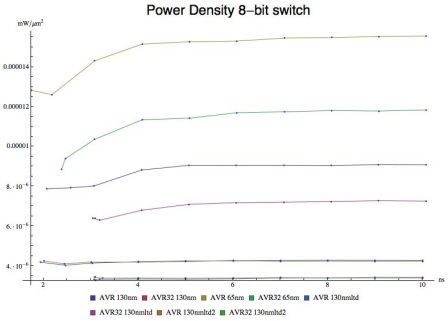


Figure 4.8: Power Density Graphs for the 8-bit Switch Test

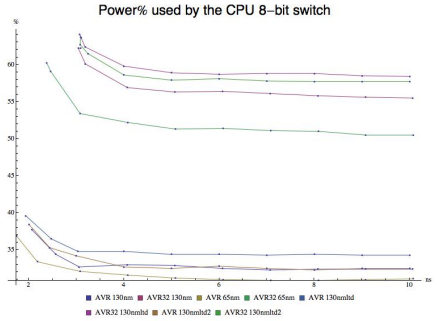


Figure 4.9: CPU % Power Graphs for the 8-bit Switch Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	1.88	1.64	1.72	3.39	2.81	2.98
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	1.89	1.73	1.77	3.39	2.84	3.04
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	1.91	1.75	1.79	3.39	3.01	3.17
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	1.81	1.55	1.61	3.19	2.52	2.67
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.05	1.09	1.17	1.08	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.07	1.04	1.06	1.12	1.06	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.4	0.36	0.38	0.38	0.32	0.35
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.08	1.04	1.06	1.15	1.09	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.04	1.	1.01	1.08	0.99	1.02
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.42	0.41	0.41	0.41	0.39	0.39

Table 4.6: Power Comparisons 8-bit Switch Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	1.08	0.95	0.99	1.96	1.62	1.72
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	1.09	1.	1.02	1.95	1.64	1.75
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	1.1	1.01	1.03	1.95	1.73	1.83
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	1.04	0.89	0.93	1.84	1.45	1.54
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.05	1.09	1.17	1.08	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.07	1.04	1.06	1.12	1.06	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.4	0.36	0.38	0.38	0.32	0.35
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.08	1.04	1.06	1.15	1.09	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.04	1.	1.01	1.08	0.99	1.02
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.42	0.41	0.41	0.41	0.39	0.39

Table 4.7: Energy Comparisons 8-bit Switch Test

Figure 4.6a it is observed that the power consumption for the systems using the AVR for the 130nmltd2 and 130nm cases. Figure 4.6b shows that the all the AVR based on the 130nm library consumes more or less the power than the AVR32 65nm. Figure 4.7a shows that all systems based on the 130nm library the differences on energy consumption are small or null around the 4ns target, the same effect is observed around the 3ns mark for the systems using the 65nm library. Figure 4.3b shows the same behavior described for the power consumed by the cpus expenditure is only 80% (on average) the energy used by the AVR cores. Figure 4.9 shows two groups the ones based on the AVR32 architecture and the ones based on the AVR architecture. It is interesting to note that for the AVR 130nm core fraction being near 3ns smaller than the one near the 4ns and the AVR32 65nm is significant less.

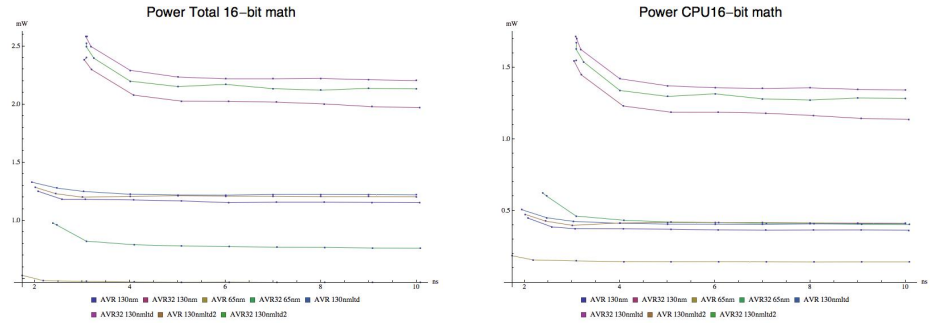
4.2.2.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	788.8	1917.6	1931.2	2028.1
NTV	—	—	181.533	190.641
STV	—	—	—	146.023
AVR32				
Super TV	705.6	1815.94	1949.22	2019.78
NTV	—	—	183.227	189.859
STV	—	—	—	145.424

Table 4.8: 8-bit Switch Energy Estimation for NTV and STV

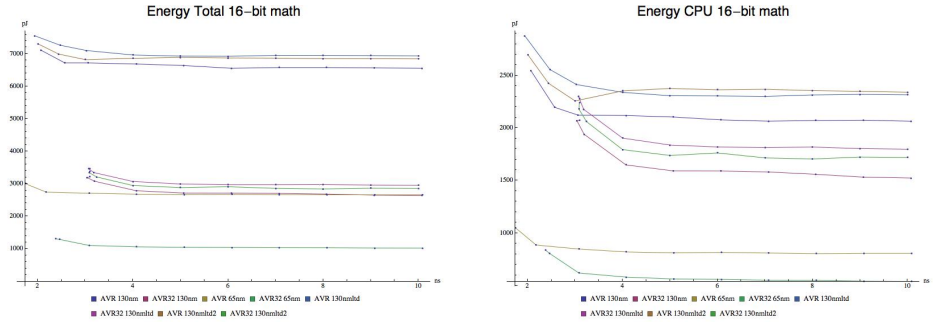
4.2.3 16-bit Math Test

4.2.3.1 Super Threshold Voltage



(a) **(b)**

Figure 4.10: Power Graphs for the 16-bit Math Test



(a) **(b)**

Figure 4.11: Energy Graphs for the 16-bit Math Test

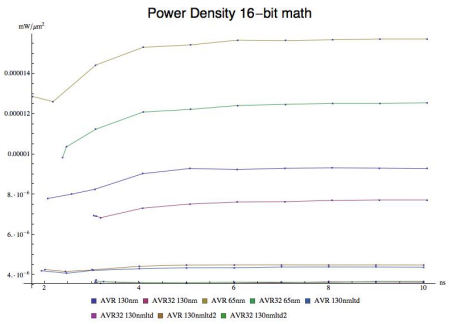


Figure 4.12: Power Density Graphs for the 16-bit Math Test

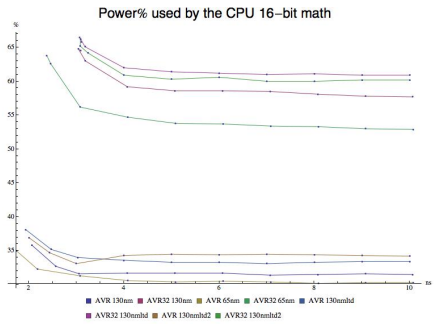


Figure 4.13: CPU % Power Graphs for the 16-bit Math Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	2.01	1.71	1.8	3.99	3.13	3.38
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	2.02	1.81	1.87	3.82	3.29	3.44
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	2.03	1.76	1.84	3.87	3.07	3.31
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	1.99	1.63	1.71	3.86	2.84	3.06
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.12	1.08	1.1	1.18	1.1	1.15
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.04	1.06	1.13	1.05	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.36	0.39	0.4	0.32	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.08	1.04	1.06	1.16	1.1	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.05	1.02	1.04	1.15	1.06	1.12
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.42	0.4	0.41	0.41	0.39	0.39

Table 4.9: Power Comparisons 16-bit Math Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	0.48	0.4	0.43	0.94	0.74	0.8
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	0.48	0.43	0.44	0.9	0.78	0.81
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	0.48	0.41	0.44	0.91	0.72	0.78
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	0.47	0.38	0.4	0.91	0.67	0.72
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.12	1.08	1.1	1.18	1.1	1.15
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.04	1.06	1.13	1.05	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.36	0.39	0.4	0.32	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.08	1.04	1.06	1.16	1.1	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.05	1.02	1.04	1.15	1.06	1.12
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.42	0.4	0.41	0.41	0.39	0.39

Table 4.10: Energy Comparisons 16-bit Math Test

In Figure 4.10a shows a group for the AVR cores and another for the AVR32 under the 130nm library. The same effect is noted on Figure 4.10b with the AVR32 65nm on the same group as the AVR cores based on 130nm library Figure 4.11a exhibits the same grouping behavior described for the power case with the note that the AVR 65nm system is clustered with the systems with AVR32 130nm.Figure 4.3b displays that if we compare just cores AVR32 cores energy expenditure is only 80% (on average) the energy used by the AVR cores. Figure 4.5Timing restriction around 3ns produces AVR core that the smallest power fraction under the 130nmltd2 case.

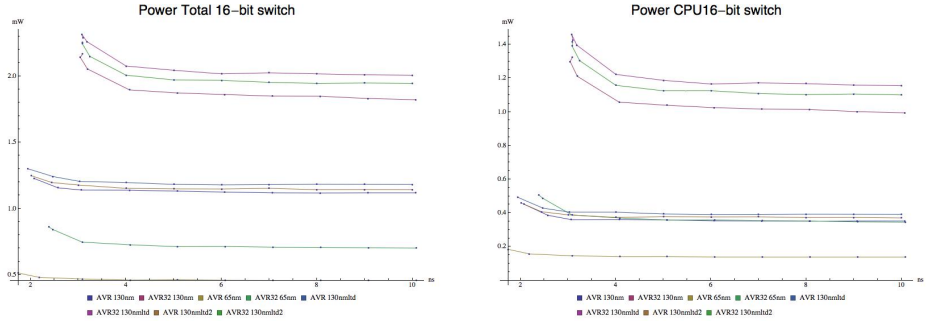
4.2.3.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	2663.92	6554.72	6844.4	6935.28
NTV	—	—	643.374	651.916
STV	—	—	—	499.34
AVR32				
Super TV	1022.42	2641.14	2856.88	2953.36
NTV	—	—	268.547	277.616
STV	—	—	—	212.642

Table 4.11: 16-bit Math Energy Estimation for NTV and STV

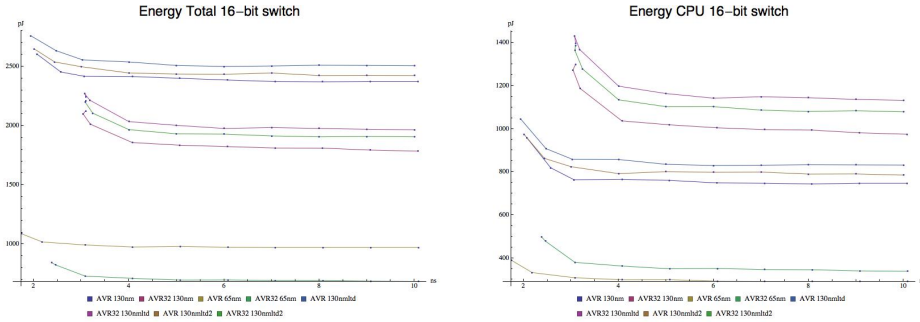
4.2.4 16-bit Switch Test

4.2.4.1 Super Threshold Voltage



(a) **(b)**

Figure 4.14: Power Graphs for the 16-bit Switch Test



(a) **(b)**

Figure 4.15: Energy Graphs for the 16-bit Switch Test

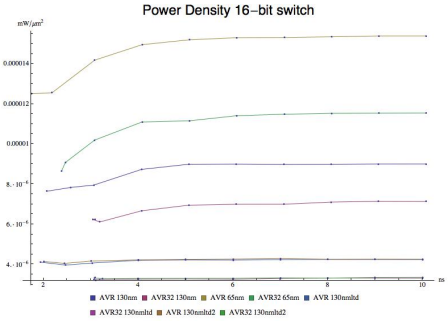


Figure 4.16: Power Density Graphs for the 16-bit Switch Test

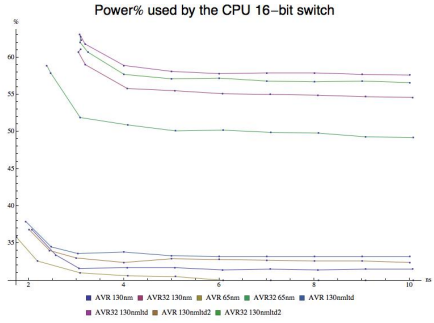


Figure 4.17: CPU % Power Graphs for the 16-bit Switch Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	1.85	1.63	1.7	3.36	2.82	2.98
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	1.88	1.7	1.75	3.45	2.89	3.06
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	1.88	1.69	1.75	3.42	2.94	3.08
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	1.75	1.53	1.58	3.11	2.51	2.64
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.1	1.06	1.09	1.16	1.08	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.07	1.04	1.05	1.11	1.07	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.4	0.36	0.38	0.38	0.32	0.35
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.12	1.09	1.11
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.03	1.01	1.02	1.08	1.02	1.05
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.42	0.4	0.41	0.41	0.39	0.4

Table 4.12: Power Comparisons 16-bit Switch Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	0.86	0.75	0.78	1.56	1.3	1.38
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	0.87	0.78	0.81	1.59	1.34	1.42
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	0.87	0.78	0.81	1.58	1.36	1.42
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	0.81	0.71	0.73	1.44	1.16	1.22
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.1	1.06	1.09	1.16	1.08	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.07	1.04	1.05	1.11	1.07	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.4	0.36	0.38	0.38	0.32	0.35
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.12	1.09	1.11
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.03	1.01	1.02	1.08	1.02	1.05
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.42	0.4	0.41	0.41	0.39	0.4

Table 4.13: Energy Comparisons 16-bit Switch Test

The case for 16-bit math and 16-bit switch have very similar results. The only thing that strikes out is that the energy consumption for both the system and the CPU is smaller when using an AVR

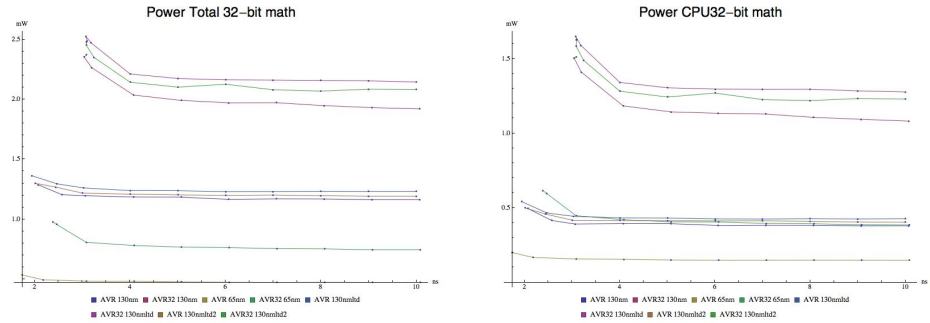
4.2.4.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	973.08	2372.28	2423.16	2505.84
NTV	—	—	227.777	235.549
STV	—	—	—	180.42
AVR32				
Super TV	687.96	1785.56	1906.1	1965.88
NTV	—	—	179.173	184.793
STV	—	—	—	141.543

Table 4.14: 16-bit Switch Energy Estimation for NTV and STV

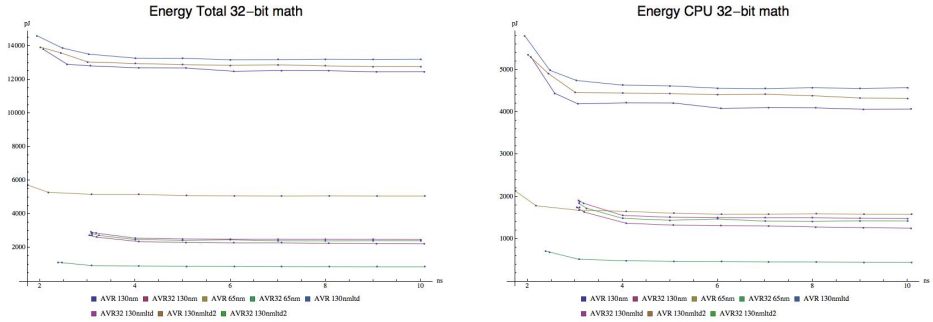
4.2.5 32-bit Math Test

4.2.5.1 Super Threshold Voltage



(a) **(b)**

Figure 4.18: Power Graphs for the 32-bit Math Test



(a) **(b)**

Figure 4.19: Energy Graphs for the 32-bit Math Test

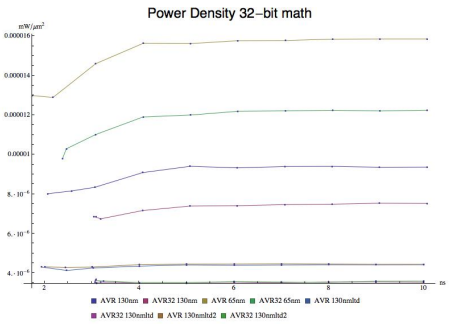


Figure 4.20: Power Density Graphs for the 32-bit Math Test

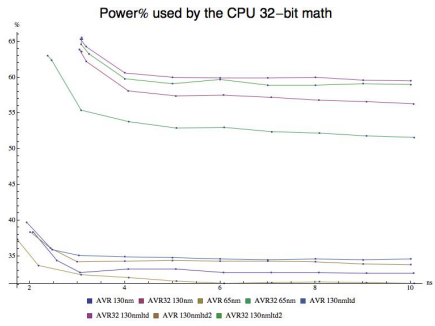


Figure 4.21: CPU % Power Graphs for the 32-bit Math Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	1.95	1.65	1.75	3.63	2.85	3.08
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	1.96	1.74	1.8	3.59	2.99	3.14
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	1.94	1.73	1.8	3.57	2.97	3.15
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	1.94	1.58	1.66	3.58	2.6	2.82
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.12	1.05	1.09	1.18	1.08	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.04	1.06	1.14	1.05	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.36	0.39	0.41	0.32	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.13	1.09	1.11
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.05	1.01	1.02	1.1	1.01	1.06
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.41	0.4	0.41	0.4	0.38	0.39

Table 4.15: Power Comparisons 32-bit Math Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	0.21	0.18	0.19	0.39	0.31	0.33
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	0.21	0.19	0.2	0.39	0.32	0.34
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	0.21	0.19	0.2	0.39	0.32	0.34
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	0.21	0.17	0.18	0.39	0.28	0.31
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.12	1.05	1.09	1.18	1.08	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.04	1.06	1.14	1.05	1.09
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.36	0.39	0.41	0.32	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.13	1.09	1.11
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.05	1.01	1.02	1.1	1.01	1.06
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.41	0.4	0.41	0.4	0.38	0.39

Table 4.16: Energy Comparisons 32-bit Math Test

For this case Figure 4.18b reveals that the implementations of the AVR on the 130nm library consume roughly the same power as the AVR32 65nm while the Figure 4.19b shows that the energy consumption of the AVR 65nm is roughly the same as the AVR32s based on the 130nm library.At the same time Figure 4.19a show that the AVR 65nm as a system consumes double the energy as the 130nm library based AVR32.

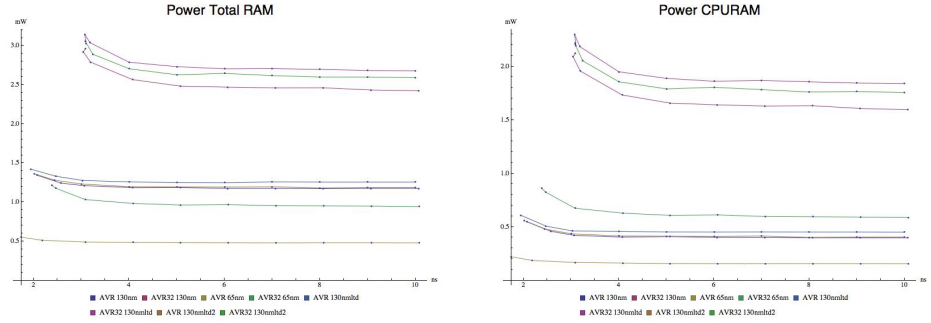
4.2.5.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	5070.56	12478.1	12778.2	13217.8
NTV	—	—	1201.15	1242.47
STV	—	—	—	951.679
AVR32				
Super TV	864.2	2228.36	2417.44	2489.36
NTV	—	—	227.239	234.
STV	—	—	—	179.234

Table 4.17: 32-bit Math Energy Estimation for NTV and STV

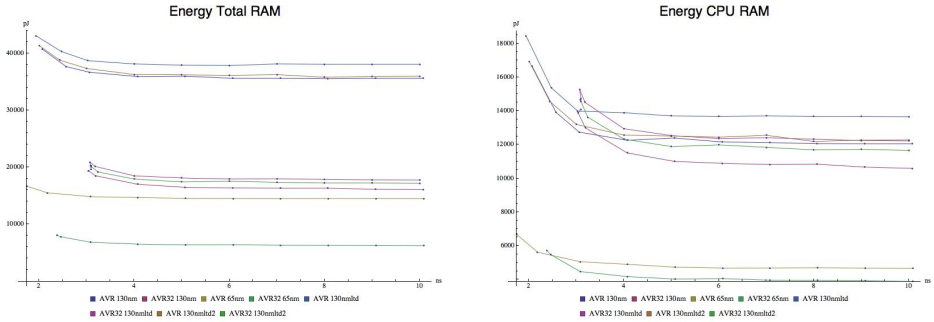
4.2.6 RAM Write Test

4.2.6.1 Super Threshold Voltage



(a) **(b)**

Figure 4.22: Power Graphs for the RAM Test



(a) **(b)**

Figure 4.23: Energy Graphs for the RAM Test

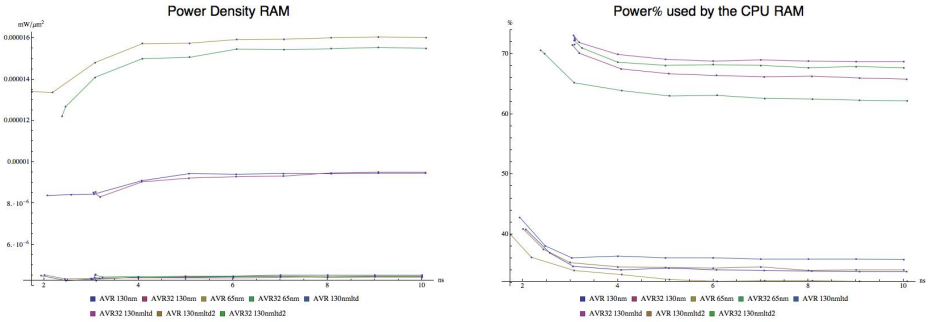


Figure 4.24: Power Density Graphs for the RAM Test

Figure 4.25: CPU % Power Graphs for the RAM Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	2.35	2.06	2.16	4.66	3.87	4.18
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	2.38	2.13	2.2	4.74	3.6	4.19
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	2.39	2.18	2.24	4.72	3.98	4.39
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	2.3	1.97	2.06	4.45	3.81	3.95
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.1	1.02	1.09	1.15	1.03	1.12
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.07	1.03	1.06	1.1	1.04	1.08
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.37	0.39	0.4	0.34	0.37
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.14	1.1	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.03	1.01	1.01	1.05	1.01	1.02
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.41	0.4	0.41	0.4	0.38	0.39

Table 4.18: Power Comparisons RAM Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	0.51	0.45	0.47	1.02	0.85	0.91
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	0.52	0.47	0.48	1.04	0.79	0.92
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	0.52	0.48	0.49	1.03	0.87	0.96
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	0.5	0.43	0.45	0.97	0.83	0.86
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.1	1.02	1.09	1.15	1.03	1.12
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.07	1.03	1.06	1.1	1.04	1.08
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.41	0.37	0.39	0.4	0.34	0.37
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.14	1.1	1.12
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.03	1.01	1.01	1.05	1.01	1.02
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.41	0.4	0.41	0.4	0.38	0.39

Table 4.19: Energy Comparisons RAM Test

In the case of RAM test Figure 4.22a shows that the AVR 130nmltd2 system, is very close to the AVR 130nm one. It stands out that the AVR32 65nm system is close to the AVRs130nm based Figure 4.22b show us a larger consumption by the AVR32 65nm core than any of the AVRs. On the energy side the AVR 65nm system consume slightly less energy than any of the AVR32 130nm spawned systems as seen on the Figure 4.23a. Meanwhile Figure 4.23b show us a more complex picture where the AVR32 130nm based CPUs consume less energy than the AVR 130nm ones above the 5ns target. The Figure 4.24 shows three distinct on the top the 65nm based systems followed by the AVR and AVR32 130nm -the ones that us the full library- and the rest at the bottom. Figure 4.25 shows two groups the AVR32 on the top between 62% and 72% and the AVR between 38% and 41%

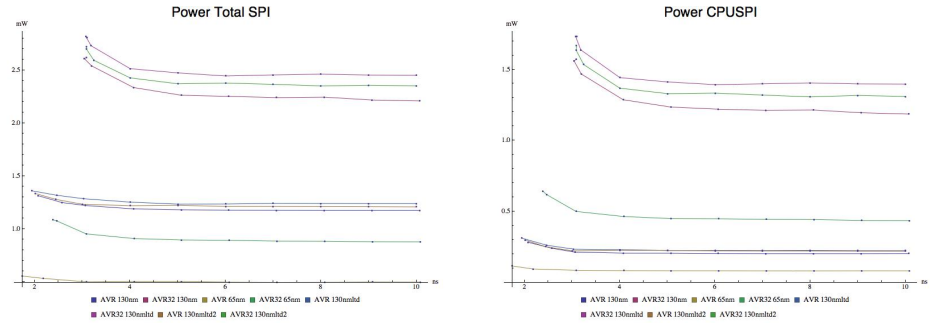
4.2.6.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	14502.5	35649.5	36013.6	38107.
NTV	—	—	3385.28	3582.06
STV	—	—	—	2743.71
AVR32				
Super TV	6261.52	16095.4	17210.9	17781.9
NTV	—	—	1617.82	1671.5
STV	—	—	—	1280.3

Table 4.20: RAM Energy Estimation for NTV and STV

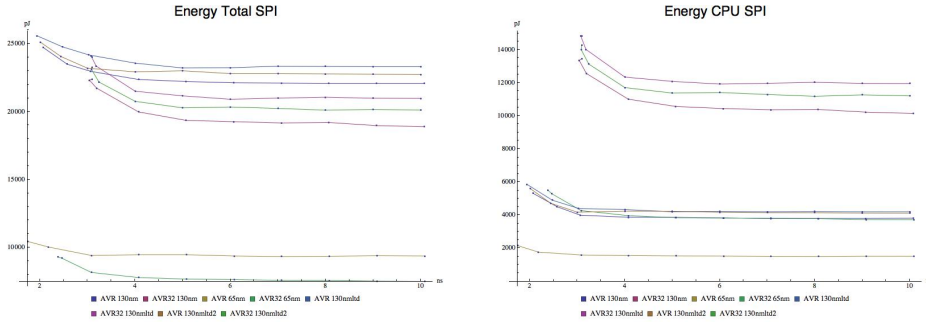
4.2.7 SPI Read and Wirte Test

4.2.7.1 Super Threshold Voltage



(a) **(b)**

Figure 4.26: Power Graphs for the SPI Test



(a) **(b)**

Figure 4.27: Energy Graphs for the SPI Test

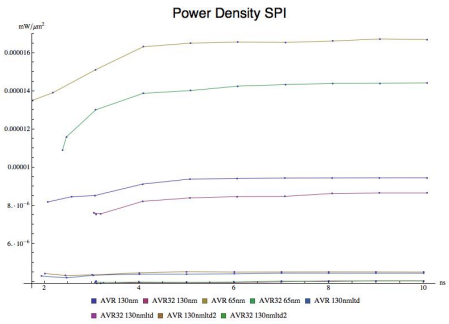


Figure 4.28: Power Density Graphs for the SPI Test

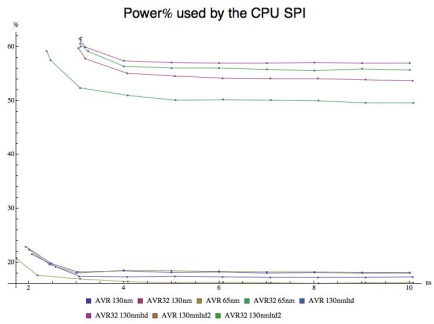


Figure 4.29: CPU % Power Graphs for the SPI Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	2.09	1.88	1.95	6.91	5.57	6.1
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	2.14	1.98	2.02	7.	5.57	6.31
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	2.11	1.94	1.99	6.91	5.6	6.09
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	2.02	1.76	1.83	6.6	5.39	5.66
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.07	1.09	1.18	1.1	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.06	1.02	1.05	1.1	1.05	1.08
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.42	0.38	0.4	0.41	0.34	0.37
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.06	1.04	1.05	1.12	1.09	1.1
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.04	1.01	1.03	1.1	1.04	1.08
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.43	0.41	0.42	0.41	0.39	0.4

Table 4.21: Power Comparisons SPI Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	0.95	0.86	0.89	3.14	2.53	2.77
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	0.97	0.9	0.92	3.19	2.53	2.87
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	0.96	0.88	0.91	3.14	2.55	2.77
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	0.92	0.8	0.83	3.	2.45	2.57
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.07	1.09	1.18	1.1	1.14
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.06	1.02	1.05	1.1	1.05	1.08
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.42	0.38	0.4	0.41	0.34	0.37
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.06	1.04	1.05	1.12	1.09	1.1
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.04	1.01	1.03	1.1	1.04	1.08
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.43	0.41	0.42	0.41	0.39	0.4

Table 4.22: Energy Comparisons SPI Test

The SPI power figures present the same case a similar behavior between the AVRs on the 130nm library and the AVR32 65nm.For the total system power expenditure shows that AVR32 above the 3ns target synthesis periods it consumes energy as shown by Figure 4.27a. Figure 4.27b Shows two groups the AVR 130nm based and AVR32 65nm on the bottom and the AVR32 130nm cores no the top.

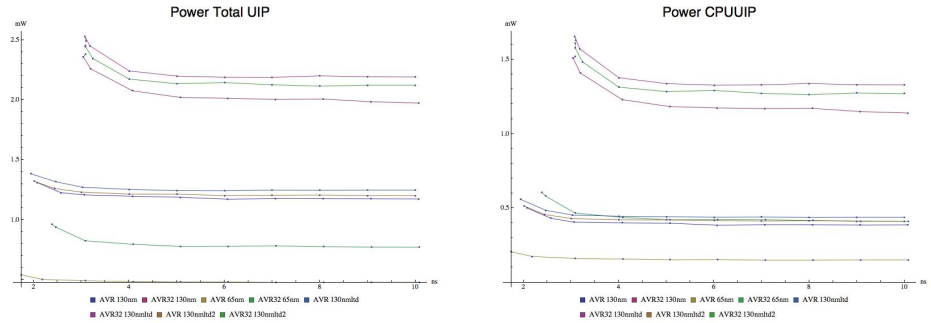
4.2.7.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	9372.36	22094.7	22734.6	23318.
NTV	—	—	2137.05	2191.89
STV	—	—	—	1678.89
AVR32				
Super TV	7507.12	18900.5	20124.6	20980.6
NTV	—	—	1891.71	1972.17
STV	—	—	—	1510.6

Table 4.23: SPI Energy Estimation for NTV and STV

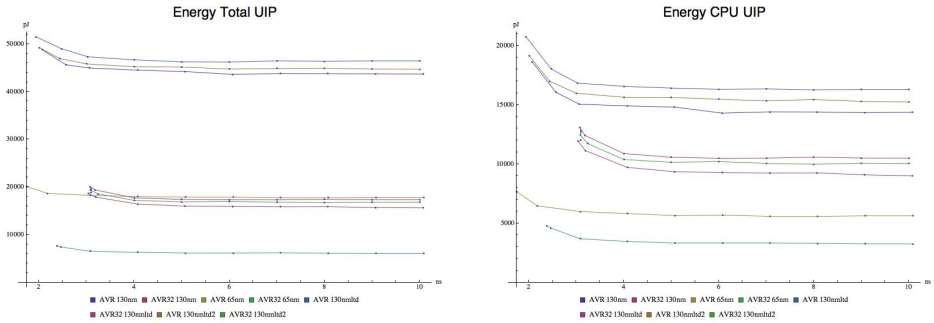
4.2.8 TCP/IP Checksum Computation Test

4.2.8.1 Super Threshold Voltage



(a) **(b)**

Figure 4.30: Power Graphs for the TCP/IP Checksum Test



(a) **(b)**

Figure 4.31: Energy Graphs for the TCP/IP Checksum Test

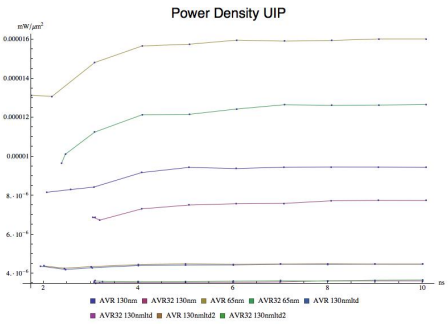


Figure 4.32: Power Density Graphs for the TCP/IP Checksum Test

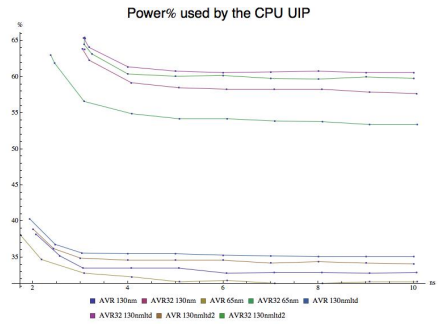


Figure 4.33: CPU % Power Graphs for the TCP/IP Checksum Test

	Total Power			CPU Power		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	1.93	1.68	1.76	3.5	2.95	3.11
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	1.93	1.75	1.8	3.48	2.92	3.11
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	1.94	1.75	1.81	3.46	3.04	3.17
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	1.87	1.61	1.67	3.34	2.72	2.86
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.05	1.09	1.17	1.07	1.13
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.03	1.05	1.11	1.05	1.08
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.4	0.36	0.39	0.4	0.33	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.14	1.11	1.13
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.03	1.01	1.02	1.08	1.03	1.06
$(\frac{\text{AVR65}}{\text{AVR130}})$nm	0.41	0.4	0.41	0.41	0.38	0.39

Table 4.24: Power Comparisons TCP/IP Checksum Test

	Total Energy			CPU Energy		
	Max	Min	Mean	Max	Min	Mean
$(\frac{\text{AVR32}}{\text{AVR}})$130nm	0.41	0.36	0.37	0.74	0.63	0.66
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd	0.41	0.37	0.38	0.74	0.62	0.66
$(\frac{\text{AVR32}}{\text{AVR}})$130nmltd2	0.41	0.37	0.38	0.73	0.65	0.67
$(\frac{\text{AVR32}}{\text{AVR}})$65nm	0.4	0.34	0.35	0.71	0.58	0.61
$(\frac{\text{AVR32ltd}}{\text{AVR32}})$130nm	1.11	1.05	1.09	1.17	1.07	1.13
$(\frac{\text{AVR32ltd2}}{\text{AVR32}})$130nm	1.08	1.03	1.05	1.11	1.05	1.08
$(\frac{\text{AVR3265}}{\text{AVR32130}})$nm	0.4	0.36	0.39	0.4	0.33	0.36
$(\frac{\text{AVRltd}}{\text{AVR}})$130nm	1.07	1.05	1.06	1.14	1.11	1.13
$(\frac{\text{AVRltd2}}{\text{AVR}})$130nm	1.03	1.01	1.02	1.08	1.03	1.06
$(\frac{\text{AVR65}}{\text{AVR132}})$nm	0.41	0.4	0.41	0.41	0.38	0.39

Table 4.25: Energy Comparisons TCP/IP Checksum Test

Figure 4.30a and Figure 4.30b exibits the same behavior Figure 4.18a and Figure 4.18b. Figure 4.31a shows that the energy consumed by the AVR 65nm system consumes more or less the same oiler as the AVR32 130nm library based systems. Figure 4.31b exposes that the AVR32 cores consume less power than the AVR ones -all form the 130nm library-

4.2.8.2 NTV and STV

(pJ)	65nm	130nm	130 ltd2	130 ltd
AVR				
Super TV	17791.2	43696.3	44701.2	46450.6
NTV	—	—	4201.91	4366.35
STV	—	—	—	3344.44
AVR32				
Super TV	6083.	15610.4	16787.5	17324.7
NTV	—	—	1578.03	1628.52
STV	—	—	—	1247.38

Table 4.26: TCP/IP Checksum Energy Estimation for NTV and STV

4.2.9 Summary

For the super- threshold voltage operation it has been shown the power consumption of the AVR32 core is form 2.4 to 7 times (been 3.7 the mean) than the AVR core but the energy spent by the AVR32 core is from 0.3 to 3.2 times (been 1.2 the mean) the AVR. At the same time the total power consumed by the system -core, bus and peripherals- when using the AVR32 form 1.5 to 2.4 times larger (been 1.8 the mean) while in terms of energy expenditure when using the AVR32 core is form 0.2 to 1 times (been 0.6 the mean) times the AVR.

For the STV and NTV estimations it is shown that is more effective to reduce de voltage than scaling to the 65nm library. For the NTV case using the 130ltd2 library is more effective than using the 130ltd.

Chapter 5
Discussion

This chapter is decided into four sections to point out the most interesting parts in comprehensible way. First synthesis process is discussed, then the results for the power and energy experiments,after that there is some suggestions along with some things I wanted to try and at the end the final conclusions are presented.

5.1 Synthesis

The synthesis was a very important part of this project. Overall the work on this area was sufficient for running the power and energy experiments. The reports generated provided useful insight to the net lists delivered by the synthesis tool. The libraries provided were good for the initial approach to get insight on this how the different parameters affect the energy and power. It is worth to remember that most of the algorithms that are used for the synthesis are non exact and heuristics since the problems solved by the tool are NP problems. It is very unlikely that the synthesis find an optimal solution. Particularly intersting is the case of the net list for the AVR 130nmltd2 targeting 3ns that proves to be more efficient than other more relaxed net-list.

5.1.1 Area

The results of the area were as expected, systems based on larger cores took a larger area, smaller feature size library lead to smaller footprint. Restricting the library forced the tool to find a larger solution and targeting a smaller clock period ended in a larger net list as less logic depth is allowed until some point is reached in where the area increases a lot but the timing is not met.

5.1.2 Timing

The synthesis tool was able to hit the targeted clock period with in some error until it was so constraint that no solution was found and for some synthesis the achieved

time is larger than a previous solution and takes a larger area.

5.2 Power and Energy

Several experiments have been done on this work to watch the influence on power and energy. Two cores were tested, different libraries were used and different conditions for a library were imposed to make it more suitable for a STV and NTV. The system is very limited in terms of number of peripherals and small memory. This emphases the roll that the CPU has on the overall amount of power and energy used.

5.2.1 AVR vs AVR32

The heterogeneous architectures seen in Section 2.2 the cores are similar. For the *Kal-El* they have different implementation of the same core and for the *big.LITTLE* the same piece of code can be executed on both cores. While on this thesis the same C code had to be complied, assembled and linked by a set of tools of the AVR and AVR32.producing executable code that can not be executed on the that was not intended to. On top of that the AVR is an 8-bit CPU and AVR32 is a 32-bit CPU. If this is not different enough on top of that they interface with the system on different busses adding complexity to the system and motivates to find out which tasks are more energy efficient to be done on one core or another. All this differences en up boiling in to three main contributing factors. Data width, execution time a task takes and power consumed by the computation.

The AVR can only perform 8-bit operations while the AVR32 can perform 32-bit operations. This mean that the AVR will require several operations to do a 32-bit math operation which is a clear disadvantage if there are a lot of 32-bit operations. But not the other if the operations are 8 bit the hardware of the AVR32 is too large and still consumes power and energy even if no useful work is done.

The AVR32 consumes more power than the AVR which might give us the idea that on the long run the AVR32 will be more expensive to operate. That is particularly good for the system is limited in power. Power can be limited because the power source is limited or there is a constraint on the amount of heat that can be removed.

Execution time is very important for real time applications where knowing how much time a task will take is needed to guarantee the task will meet the deadline. But also is important because is needed to compute the total energy performing a task will take. It is possible that a CPU requiring higher power but requiring less time use less energy for solving the same task as a low power CPU that takes a longer time to finish the task. As a matter of fact this is the main reason why the systems based on the on the AVR32 use less energy for competing the tests than

the AVR except for the cases were the difference of the execution time is not large enough to counter the extra expenditure on power like 8-bit switch test. In which the AVR just takes 1.73 times the time than for the AVR32.

5.2.2 Core vs System

As seen with the test cases for SPI, 16-bit switch and RAM the CPU energy expenditure does not explains why the AVR32 based systems use less energy than the AVR based systems use less energy on the task overall. That is the rest of the peripherals contribution. Even on the RAM test case -the one where the CPUs use the maximum energy share up to 70%- the system contribution is enough to make the AVR32 systems use less energy. It is theoretically possible to have a large enough system where the CPU choice does not make a difference on the power, and energy profile. As a matter of fact if the peripherals were turned of during the RAM test the systems based on the AVR would have been the more energy efficient.

5.2.3 NTV and STV

If the assumptions stated on Section 3.3.2 apply choosing least energy expenditure implementation of the core then depends on the voltage management philosophy. If it is STV and it is fixed then the 130nmltd is the best alternative. If for some reason dynmaic voltage is needed (to adjust performance to work load) and the STV is not needed then the best way to go is with the 130nmltd2 approach. If the system is limited to super-threshold voltage then 130nm (without restrictions) will provide the lowest energy/power foot print.

5.3 Future Work

There are a lot of interesting ways to extend the range of this project. To have a better picture on how better to adapt to different conditions that might present in different applications.

5.3.1 Test Cases and Peripherals

More cases must be added and profiled. some of them might be memories with wait states, burst transactions shared caches, multiple masters tying to access a salve simultaneously, interrupts, DMAs, going to/ waking form sleep states, sleep states, watchdog routines, encryption etc. On the peripheries side USART, bluetooth, ethernet, USB and ADCs, RFID readers are a good place to start to get a broader picture on the huge set of applications useful under the IoT scope.

Other interesting thing will be to include the ability to turn on or off different peripherals to quantify the contributions of them on the system and ultimately will end giving the final application better control on the energy and power expenditure.

5.3.2 Real STV, NTV and More Libraries

As stated earlier there was no STV nor NTV library available for this project. Performing the synthesis on such libraries will yield to more accurate results and provide a more detail picture on the effects on power, leakage and timing. The use of other libraries featuring different sizes will provide a better understanding on the effects of seizing.

5.3.3 Tools and Testing Cores at the Same Time

The development of tools that address both cores at the same time will speed up the generation and testing of cases. An interesting idea is to be able to target an optimization target like performance, energy, power and permit the cooperation between cores.

Also it is important to understand how the handover of tasks and the address of shared resources affect the time a task takes, the power it draws, etc.

5.3.4 Cores

To test other cores (AVR AP7, megaAVR, tinyAVR and /or ARM Cortex M0+) and different set-ups will help to encounter more suitable solutions to other environments and applications.

5.4 Conclusions

This project has shown the great diversity provided by testing two different cores, in one intense we have that the AVR32 is in general terms more energy efficient than the AVR because it takes less time to do things. On the other hand if the main source of power is a harvesting system that can not provide peaks of power but a moderate supply then the AVR would be better since is easier to make it comply to the power constrain.

References

[1] Agarwal, A., Mukhopadhyay, S., Kim, C., Raychowdhury, A., and Roy, K. (2005). Leakage power analysis and reduction: models, estimation and tools. *Computers and Digital Techniques, IEE Proceedings*, 152(3):353–368.

[2] Ashton, K. (2009). That 'Internet of Things' Thing. "http://www.rfidjournal.com/articles/view?4986".

[3] Atmel (2000). Module documentation 8-bit core - avr8_i6000. *Confidental*.

[4] Atmel (2012). 8-bit Atmel XMEGA A Microcontroller. "http://www.atmel.com/Images/doc8077.pdf".

[5] Atmel (2013). 32-bit atmel avr microcontroller. "http://www.atmel.com/Images/Atmel-32145-32-bit-Flash-MCU-UCL0_datasheet.pdf".

[6] Atmel (2014). Quick RefeRence Guide. "http://www.atmel.com/Images/doc4064.pdf".

[7] Bo Zhai, D. B., Sylvester, D., and Flautner, K. (2005). Theoretical and practical limits of dynamic voltage scaling. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions*, 13(11):868–873.

[8] Chen, P.-Y., Fang, C.-C., Hwang, T., and Ma, H.-P. (2009). Leakage reduction, variation compensation using partition-based tunable body-biasing techniques. *VLSI Design, Automation and Test, 2009. VLSI-DAT '09. International Symposium on*, pages 170–173.

[9] Corporation, F. T. (2003). 0.13um high performance high density standard cells-fsc0h_d core cell. Technical report, Faraday Technology Corporation.

[10] Corporation, F. T. (2008). Fse0a_dhm_generic_core 65 nm low-power core cell library. Technical report, Faraday Technology Corporation.

[11] Dreslinski, R., Wieckowski, M., Blaauw, D., Sylvester, D., and Mudge, T. (2010). Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266.

[12] G. D. Wilk, R. M. Wallace, J. M. A. (2001). High-gate dielectrics: Current status and materials properties considerations. *Journal of Applied Physics*, 89(10):5243–5275.

[13] Greenhalgh, P. (2011). big.little processing with arm cortex-a15 & cortex-a7. "http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf".

[14] Hu, J., Shin, Y., Dhanwada, N., and Marculescu, R. (2004). Architecting voltage islands in core-based system-on-a-chip designs. *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, (August):180–185.

[15] Johnsen, G. (2014). Full-custom sub/near-threshold cell library in 130nm cmos with im- plementation to an alu. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet (NTNU).

[16] Kristian Saether, I. F. (2014). Introducing a new breed of microcontrollers for 8/16-bit applications. "http://www.atmel.com/Images/doc7926.pdf".

[17] Kumar, R., Farkas, K., Jouppi, N., Ranganathan, P., and Tullsen, D. (2003). Single-isa heterogeneous multi-core architectures: the potential for processor power reduction. *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 81–92.

[18] Limited, A. (1999). *AMBA Specification (Rev 2.0)*. ARM Limited.

[19] Limited, A. (2006). *AMBA 3 AHB-Lite Protocol v1.0 Specification*. ARM Limited.

[20] Mahmood-Qureshi, Y. (2013). Low-power optimized apb bus connection for avr co-processor. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet (NTNU).

[21] Niel, G., Raffi, K., and Cohen, D. (2004). The internet of things. *Scientific American*, 291(4):76–81.

[22] Niel H. E. Weste, D. M. H. (20011). *CMOS VLSI Design a Circuits and Systems Perspective*. Addison-Wesley.

[23] nvida (2011). Variable smp - a multi-core cpu architecture for low power and high performance. "http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911b.pdf".

[24] Robert Boylestad, L. N. (1999). *Electronic Devices and Circuit Theory*. Prentice Hall.