BENCHMARKING ARM-BASED APPLICATION

INTEGRATED SYSTEMS

By

SETH WILLIAMS

Bachelor Science in Electrical Engineering

Oklahoma State University

Stillwater, Oklahoma

2011

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE IN ELECTRICAL
ENGINEERING
December, 2011

BENCHMARKING ARM-BASED APPLICATION

INTEGRATED SYSTEMS

Thesis Approved:

Dr. James Stine

Thesis Adviser

Dr. Sohum Sohoni

Dr. Weihua Sheng

Dr. Sheryl Tucker

Dean of the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The line between a full processor and a microprocessor has always been a rather subjective one. Subsequently, there are only a few technologies straddle that line more so than Advanced RISC Machine (ARM) based applications processors [1]. These behemoth microprocessors are at the heart of almost every major mobile handset [2] and capable of running full operating systems while still maintaining the low power advantages of their embedded brethren. As these applications processors become more adept, and subsequently the devices based on them become more feature laden, they become even more difficult to define. As the evolution of these devices progress, they will continue to become a category of their own, as such it stands to reason that they should be treated as a category of their own.

The overwhelming majority of these application processors utilize the ARM Instruction Set Architecture (ISA) [2]. ARM is used in a variety of different embedded systems ranging from the high-end applications processors to low end microcontrollers. This ISA is the Intellectual Property (IP) of the company ARM Holding Ltd. The company, founded on the development of an early version of the ARM ISA, has spent decades modernizing and expanding the architecture. This has led to many innovations that have contributed to the further differentiation between ARM and other embedded ISAs. Instead of manufacturing processors themselves, they license the IP cores to other companies allowing them to modify the technology to their own needs. This arrangement has resulted in literally hundreds [3] of companies producing billions of ARM processors per year [4] [5].

Rarely is licensing and manufacturing an exact copy of a core enough to meet the needs of a complex design. Thus licensees often times modify the core, package it with other components, or both. Using space reduction packaging technologies these companies are capable of containing an entire system to one footprint on a circuit board. While there are different methods to packaging system components together, for the purposes of this paper and project they shall all be referred to as integrated systems [6] [7] [8]. These integrated systems come in many varieties of capability, function, and complexity.

It should be obvious that many of these integrated system designers are direct competitors. It stands to reason then, that just like choosing any other option of processors it is important to be able to compare and differentiate between similarly purposed integrated systems. While a plethora of benchmarks exist for embedded systems and full processors, and even a few for applications processors, finding a benchmark that tests an entire system contained within a single device is much more difficult. Even when using the same core for the same design goal, two companies can and likely will have widely divergent components within the integrated system. Likewise, the methodology of packaging the components will have differences as well. It isn't enough just to test the core, but the entire system as a whole must be benchmarked due to the components being inseparable. After all, for this purpose, it behaves as a single device.

OBJECTIVES

Because of their prevalence, the intent of this thesis is to explore the topic of benchmarking integrated systems that are specifically in the applications market and powered by ARM technology. There are several questions on this subject that should be answered. Importantly, whether these ARM-based applications processors differ enough to be treated separately should be answered. The next question is whether or not full knowledge of the core obviates the necessity of testing the full system in these cases. Along with this, determination of whether or not there is an acceptable benchmark suite that is capable of adequately testing the full capabilities of an integrated system will be equally important. If there is not, then the reasons one hasn’t been developed need to be investigated. The final consideration is what the future holds for these devices, and the part that will play.

CONTRIBUTIONS

This paper makes several contributions.

- The collection of technologies included within the boundaries of packaging technology that are dedicated to containing a system to a single footprint is shrouded in ambiguity and conflicting nomenclature. The terms system-on-chip, system-in-package, and package-on-package are frequently used in slightly different nuances, and occasionally interchangeably, while in other research instances they are heavily distinguished. This overview and explanation will help to remove this ambiguity and will clarify the all-encompassing definition of “integrated systems”.
- It is necessary to determine what the acceptable characteristics are for a standard benchmark intended for integrated systems. To do so, a brief summary of critical features that should be expected in a standard benchmark is overviewed. The topic of misusing benchmarks is also reviewed and discussed.
- The OMAP3530 [9] integrated system will be tested with the benchmark suite MiBench [10]. The results of running the benchmark are included, with thorough documentation, charts, time stamps, and other relevant information.

- A thorough exploration of the state of benchmarks applicable to the target platforms (ARM-based applications integrated system) shall be completed. This includes determining if the industry standards are sufficient as well as providing a succinct collection of previous relevant experiments and results.

SUMMARY

Competing designs of ARM-based applications integrated systems are widespread enough to warrant their own benchmarking standard. To test just the ARM cores is not sufficient to compare or contrast the capabilities of the integrated systems. Additionally application end processors balance the strengths of normal embedded microprocessors with full processors, thus benchmarking against one or the other category is inconclusive and superfluous. With the growing industry the necessity will continue to climb.

Chapter 1 provides a detailed overview of the project and the thesis statement. The second chapter focuses on the background knowledge necessary for the project. This begins with a detailed overview of the ARM Instruction Set Architecture starting with its history and prominence in modern devices and ending with specific details of the architecture. The section on ARM is followed by a look at what makes an integrated system by overviewing the specifics of some of the packaging technologies that are used to contain an entire system to one footprint. Next a detailed look at the test device itself, the Beagle Board, is included. The background chapter is completed with an overview of benchmarking. *Chapter 3* includes a close examination of preparing the Beagle Board for the experiment, a look at selecting the appropriate benchmark, and the specifics of the benchmark selected. Following this, the fourth chapter details the results of the work. This begins with an examination of the results of benchmarking the Beagle Board. The other prevalent part of this chapter looks at the results of researching various benchmarks for existing results and comparisons between system integrated circuits such as those between the OMAP and i.MX platforms. *Chapter 5* concludes the thesis.

II. BACKGROUND

In order to understand the necessity of a benchmark suite specifically aimed at integrated systems that use ARM ISAs, it is imperative to have a background. As mentioned in the Introduction, ARM is very widely licensed and its use is rapidly expanding. These licensees are using ARM based technology in different ways to create their own systems. Some of these systems are integrated together on a single chip or within a package and sold as its own product; therefore, an overview is necessary of both ARM technology and packaging techniques for integrated systems.

One such product is TI's OMAP series. The Beagle Board is a convenient interface with an OMAP processor, thus is used as a platform for benchmarking experiments. It is highly functional, and is adaptable to a large variety of projects. The Beagle Board also has the advantage of being fully open-source. This device will be further explored in the sections below.

The study of benchmarks has been thoroughly explored, so it is not the purpose of this paper to broaden or expand this topic. However, it is necessary to examine some of what is available in order to highlight the absence of applicable benchmark suites. It will become clear that there are many benchmarks that do provide some suitable tests for these systems. However, none of these benchmarks are satisfactorily comprehensive.

ARM

One of the most common Instruction Set Architectures (ISA) being developed for modern applications is the Advanced RISC Machine (ARM). ARM is a 32-bit ISA based on the Reduced Instruction Set Computer (RISC) design strategy. This architecture has made sweeping advances since its conception, expanding into multiple embedded markets, particularly those related to consumer electronics. It is important to understand how widespread ARM cores have developed and how rapidly they have achieved that level of success in addition to the processor capabilities and architecture.

History and Marketing

Originally, ARM stood for Acorn RISC Machine and was developed by a branch of a small British company, named Acorn Computers Ltd, hoping to get into the business computing market. There were not any viable processor options that fit their needs or market goals, so they chose to develop a new architecture after being inspired by a RISC project completed by a group of Berkeley graduate students proving architecture development could be done on low budget and limited facilities. After completing the ARM1 primarily as a development project in 1985 [11], eventually the ARM2 and later the ARM3 were marketed. In 1990, Apple Computer and Acorn's silicon contractor, VLSI Technology, aided in researching the next stage in ARM development [5]. These contributions lead the team to break off into its own company: Advanced RISC Machines Ltd. Eventually the company renamed itself ARM ltd (or ARM Holdings) in 1998 when it floated itself on the London stock market [5].

ARMs first embeddable RISC core in 1991 was the result from the early joint efforts between Apple, Acorn and V-tech [11]. This first embeddable core was also based off the new ARMv3 architecture, and was named the ARM6 as a result of a new core naming scheme. Over the next two decades several versions of the ISA were developed, the most recent being the ARMv7 debuting with the Cortex family in 2004 [5]. For a better visual of the architecture version as related to its processor family *Table 1* has been included. This table has been simplified; there are a variety of sub-architecture versions that distinguish the differences between the families. Likewise some families exist over the span of several architectures, using different sub-architectures during the lifetime of that family.

***Table 1*: ARM Versions and Families**

Architecture	Families
ARMv1	ARM1
ARMv2	ARM2 ARM3
ARMv3	ARM6 ARM7
ARMv4	ARM7TDMI ARM8 StrongARM ARM9TDMI
ARMv5	ARM7EJ ARM8 ARM9E ARM10E Xscale
ARMv6	ARM11 Cortex-M
ARMv7	Cortex-A Cortex-M Cortex-R

During this time ARM also developed a variety of innovations to allow more chip specializations and options. These will be further detailed in the *Features and Expansions* section, though introduced here. One of the more notable developments was Thumb, an operating state that uses a 16-bit subset of compressed ARM instructions that could be toggled on or off. A more sophisticated version of Thumb, titled Thumb-2, exists in the current architecture families. Thumb and Thumb-2 can also be used as the sole architecture, excluding the regular ARM instruction set altogether. Another innovation is Jazelle, a Java execution mode used to more efficiently execute Java byte code. Other more common, natural advancements included options of adding a floating-point unit (FPU), Digital Signal Processing (DSP) oriented designs, and multicore designs.

ARM Holdings operates on a business structure of licensing its cores out to different companies as intellectual property. This is done in lieu of manufacturing and selling individual chips themselves. As of 2011, the company is able to boast [2] over 15 billion ARM cores have been shipped and over 200 companies have an ARM license. When compared to the 1.5 billion licensed and sold as of 2005, the accelerating growth is clear to see. Currently over 95% [2] [12] of the mobile handsets use ARM technology, and even more impressively, the technology exists in over 25% of all electronic devices. It is expected that in 2011 [4] alone there will be 5 billion more IP cores sold.

Features and Expansions

As previously mentioned in the *History* section, many standard options that are commonly expected in embedded systems were developed for the ARM processors to increase the potential applications and industry competition. These came in a variety of feature sets ranging from the common capabilities such as inclusion of a floating point unit or multicore to much more specific and advanced options such as expansions to the instruction set with Thumb. Many of the features weren't integrated until more recent revisions of the ISA.

When looking at the more common features available to standard specific-purpose microprocessors, some of the most prominent recurring options include DSP and FPUs. In modern designs multicore is also a commonly available feature. A general purpose Single Instruction Multiple Data (SIMD) [13] engine for multimedia applications is also an available option in upper end processors. SIMD extensions operate transparently with the OS utilizing existing ports. A more advanced version, titled NEON [14], was developed for the ARMv7 architecture and uses wider vectors as well as featuring its own pipeline. NEON vastly outperforms the older SIMD, at least doubling its speed [14]. The final option of note is TrustZone [15], a multi-tiered infrastructure that provides a combination of software and hardware security features tightly integrated into the processor. NEON, SIMD, and TrustZone all extend the base instruction set. In the next session, Processors, a figure will illustrate which of these features are available with the various detailed processor families.

Multiple extensions exist to supplement the basic 32-bit ARM ISA: Thumb, Thumb-2, and Jazelle [16]. Thumb is a subset of common ARM instructions reduced to 16-bits. To clarify, a processor with Thumb enabled still has 32-bit wide registers and buses, it just uses smaller instructions. This is done so that when using 16-bit memory, the processor does not need to make two fetches per instruction, which would significantly reduce performance. These instructions are then decompressed during decompression. Another advantage of this system is allowing emphasis on code density when necessary. Now, in the most recent two architectures (ARMv6 & ARMv7) Thumb-2 is also available. In actuality, Thumb-2 is a stock feature of the Cortex series. Thumb-2 is a hybrid instruction set with all of the Thumb 16-bit instructions and a subset of the original ARM 32-bit instructions, designed to seamlessly use the variable instruction length. It boasts [17] [18] a 25% boost in performance over thumb and a 26% reduction in memory usage. The original 32-bit ARM instruction set can still be included with Thumb-2, in fact it even allows for more seamless transitions. In addition to the Thumb options, Jazelle [19] is an ARM extension that is focused towards Java support. It has both software and hardware components. There are now two versions of Jazelle: Direct Bytecode eXecution (DBX) and Runtime Compilation Target (RCT). The original, DBX, allowed direct execution of Java bytecode. RCT, also referred to as ThumbEE uses Just-in-Time (JIT) and Ahead-of-Time (AOT) compilation methods. ThumbEE is capable of handling a larger variety of execution environments than just Java. Due to this it is more preferred and is supported by real-time and mandatory in application driven processors in the ARMv7 architecture.

In the previous section, *History and Marketing*, a few of the older processor families include a few letters at the end of the name (see *Table 1*). These are as follows: ‘T’, ‘D’, ‘M’, ‘I’ [20] which are usually included together, as well as ‘E’ and ‘J’. These indicate specific features. The ‘T’ is fairly obvious and indicates the Thumb extension previously described. Both the ‘D’ and ‘I’ are separate debugging options, the former standing for ‘Debug mode’ which is Joint Test Action Group (JTAG) support, and the latter meaning ICE support is available. The ‘M’ is a little less straightforward and stands for multiply to indicate that the pipelines are deeper and an enhanced multiplication instruction is used. This is relative to the older processors. The DSP feature is indicated by the ‘E’ which stands for extended, and this implies all of ‘TDMI’ is included. Finally ‘J’ indicates the Jazelle extension. In the Cortex family these labels are no longer required because many of these features are assumed to be part of the product, or have been replaced or updated.

One of the differences between some processors developed by ARM is the type of memory control unit that is used. Application-specific processors also use a more advanced memory control system than the embedded processor alternatives. These are respectively identified as Memory Management Unit (MMU) [20] and Memory Protection Unit (MPU) [20]. Both of these are used for protection against unwanted accesses to system resources. The MMU also includes hardware to support virtual memory.

Processors

There are three primary processor market categorizations used at ARM [21]: Classic, Embedded, and Applications. These are categorized by the added capabilities from advanced features, as well as increasing performance and functionality. This is demonstrated by the graph shown in

Figure 1. In addition to these, there are a few specialty processors worth briefly acknowledging that exist outside of the main three categories such as the SecurCore line for security applications and FPGA target processors.

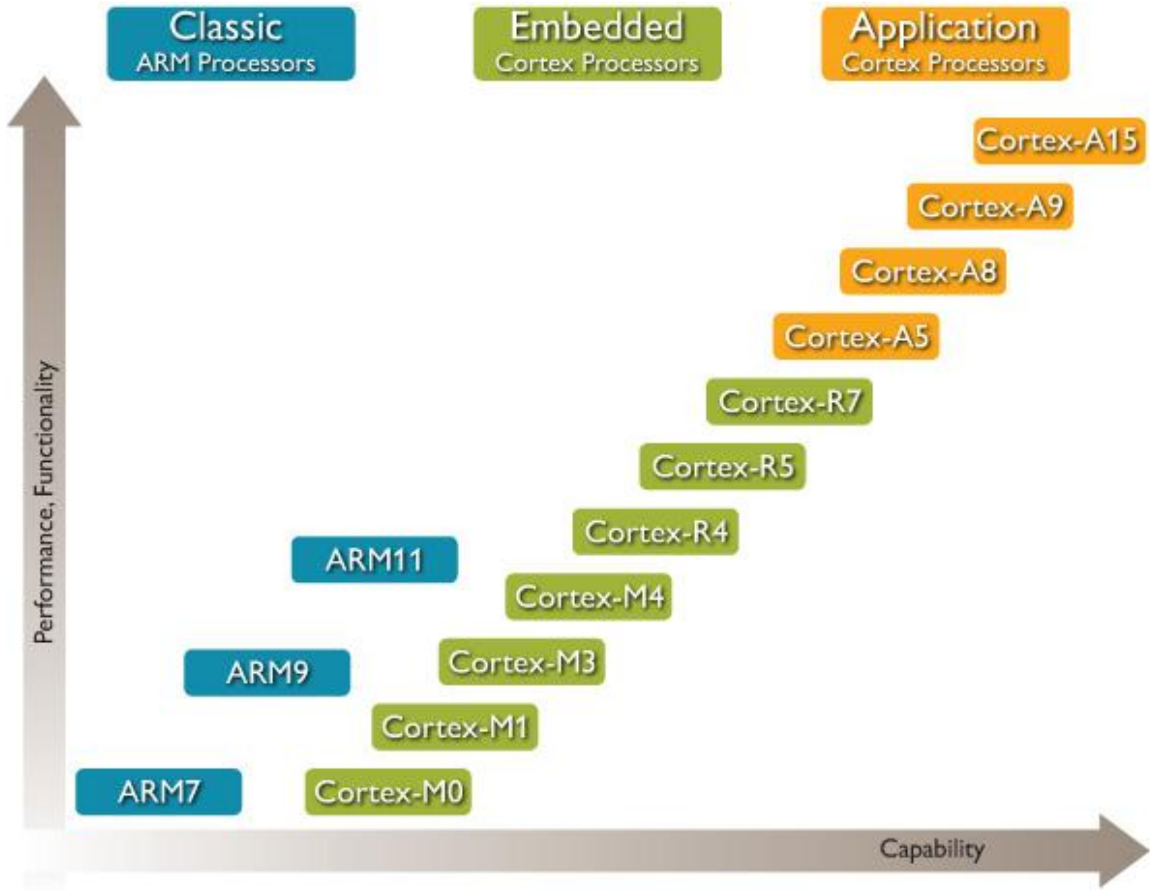


***Figure 1*: ARM Processor Categories by Capability [21]**

The classic processors consist of the previous three major ISA versions of ARM architecture. ARM7 (Actually the ARM7TDMI or ARM7EJ) [22], using ARMv4, is almost entirely antiquated at this point, with the company firmly suggesting a Cortex counterpart. ARM9 (ARMv5) [23]is still in use as a low-end single processor for DSP and java applications. ARM11 [24] is based on ARMv6, and is still seeing wide use as a potential option in modern development. Several of these older processors have a binary compatible counterpart in the Cortex family to allow for design upgrades that do not require large scale software redesign.

The second classification of processors used by ARM is the modern embedded processors, denoted Cortex-R and Cortex-M for real-time and microcontroller oriented applications respectively. Each of these utilizes the ARMv7, as indicated by the Cortex title, and thus includes the Thumb-2 Instruction set automatically along with other Cortex series standards. However to distinguish them from the applications line, both of these processor families utilize the MPU for memory control. They also operate on a Real-Time Operating System (RTOS) in conjunction with user generated code. The Cortex-R [25] features deeper pipelines and uses high clock frequencies. It also utilizes Tightly-Coupled Memory (TCM) for fast access to important data or instructions that are needed for immediate access. TCM is considered level 1 memory, and in some cores it entirely replaces the cache. In contrast, the Cortex-M [26] is designed with low-power, code density, and interruption management as focus points. The Cortex-M series exclusively uses Thumb-2 and does not have the ARM instruction set. Thumb-2 allows it to maintain the low impact design requirements of its 8/16-bit competitors while still keeping the performance advantage offered a 32-bit machine. Due to this instruction set it is able to function as the industry standard by vastly outperforming competition in a MIPs per MHZ comparison.

The final classification of processors at ARM is the applications series, Cortex-A [27]. These are used for high functionality, and are defined by their ability to run complex and complete operating systems. Differing from the embedded classification Cortex processors, the applications series uses the MMU instead of the MPU for memory control. Additionally the option of up to four cores is available supporting a fully coherent L1 cache. The Cortex-A family is more open in the number of available options and extensions than its counterparts. Certain features that are used as options in the other processor families are automatically included in all Cortex-A processors, namely Jazelle and NEON.

A more complete observation of the different features that were detailed in the prior section and their availabilities for the different processor families may be observed in *Figure 2*. This image is organized by the specific architecture used to create the columns. The top half uses color to indicate the processor classification, and the processors are listed above their respective architecture version. Listed below each of these architectures are the various options available to the specific architecture.

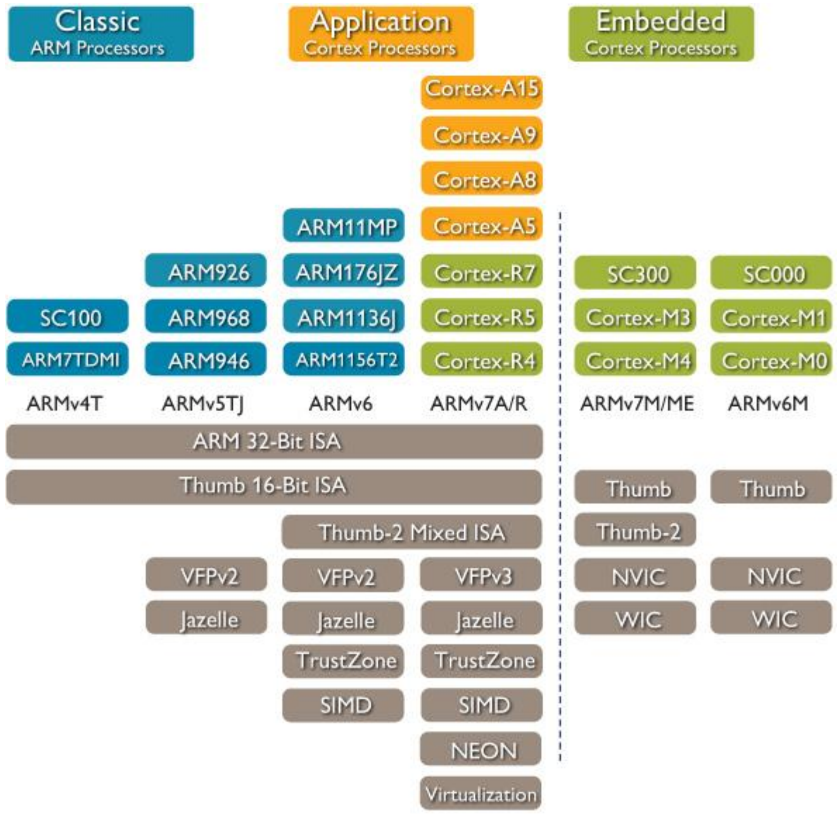


Figure 2: **ARM Processors and Features [21]**

For simplicity and disambiguation, *Table 2* is also included to specifically examine the available instruction set extensions available by each processor family. Thumb, a staple of ARM processors since its conception, is available in all models. The newer Thumb-2 is a primary feature in the more recent families. Jazelle shows itself to be available in the higher end applications models, so was excluded in the Cortex-M and Cortex-R, and wasn't available yet in the design of the ARM7. As mentioned before, the ARM ISA is completely excluded in exchange for only using Thumb-2 in the Cortex-M. Also, a slight error in *Figure 2* claims NEON is available in Cortex-R, though a closer look at the feature [14] disproves that claim.

***Table 2*: Instruction Set Options**

	ARM	Jazelle	Thumb	Thumb-2	SIMD	NEON	TrustZone	KEY
Cortex-A	R	R	R	R		R	R	
Cortex-M			R	R				R - Required
Cortex-R	R		R	R	O		O	
ARM11	R	O	R	O	O		O	O - Optional
ARM9	R	O	R					
ARM7	R		R					Blank - Unavailable

Architecture

Because ARM is based on RISC design, it shares all of the pertinent characteristics of a RISC instruction set. However, it was deemed necessary to enhance and expand the capabilities of a typical RISC machine. ARM still uses the fixed instruction width, load/store architecture, simple addressing modes, and uniform register files [16] common to RISC machines. The object of these additions were to create seamless improvements aimed at increasing throughput and compensating for some of the advantages CISC machines generally have. A couple examples of this include conditional execution to reduce branching overhead and the ability to load and store multiple instructions [16].

Currently ARM utilizes *37* registers broken down into *30* for general purpose, *6* status registers, and a program counter [28] [1]. This is used as a general standard, though certain processors do make slight modifications to this model. At any given time, fifteen of the general use registers are accessible in addition to the program counter and the status register. Which registers are available depends on which operating mode is being used by the processor. There are seven operating modes used by ARM, six of which are privileged with the seventh being the user mode. The first two privileged modes are entered for interrupt handling; IRQ for low-priority normal interrupts and FIQ for immediate needs interrupts [28]. Abort mode and undefined mode are used for memory access violations and unrecognized instructions respectively [1]. Supervisor mode is used for software interrupts and when the system is reset [28]. The system mode uses the exact same registers as the user mode. *Figure 3* illustrates the different modes and register swaps that accompany them.

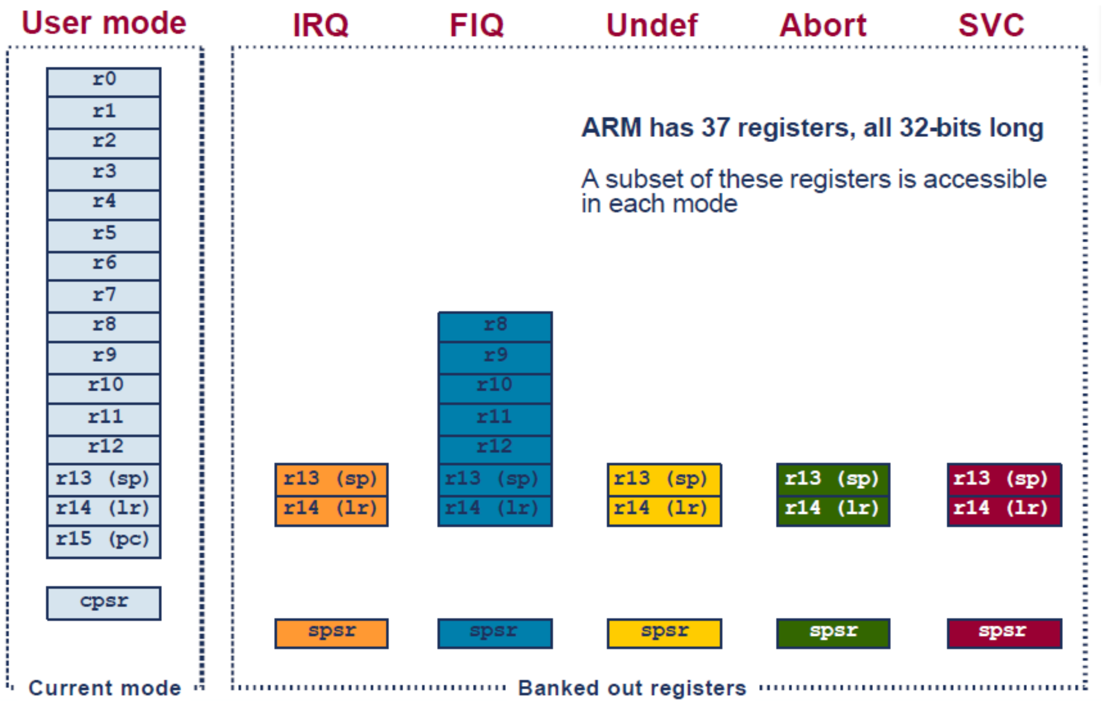


***Figure 3*: ARM Registers [1]**

The six status registers consist of a single Current Program Status Register (CPSR) and five Saved Program Status Registers (SPSRs) [28] [1]. The user and system modes make use of the CPSR, which contains the current state of the machine. Whenever the mode is changed the content of the CPSR is preserved into the corresponding SPSR. The state is stored in the SPSR to allow a return to the previous state upon completion of the interrupt or handling of the exception that prompted the mode change. The full breakdown of the program status registers can be seen in *Figure 4* below. There are a few noteworthy bits in the register. The bottom 5 bits are used to indicate the current operating mode [28]. Of high import is the 6th bit labeled T, this is a read-only bit used to determine whether or not the machine is operating in the Thumb ISA or the ARM ISA [28]. The 'I' and 'F' bits are used to enable or disable low priority and high priority interrupts respectively [28]. The 25th bit, 'J', is used to indicate if the processor is in a Jazelle state [1]. The most significant four bits are labeled NZCV and are referred to as the condition flags. These are flagged for the following conditions: negative result from ALU, result of zero from ALU, ALU operation carried out, and ALU operation overflowed [1] [16] [28].

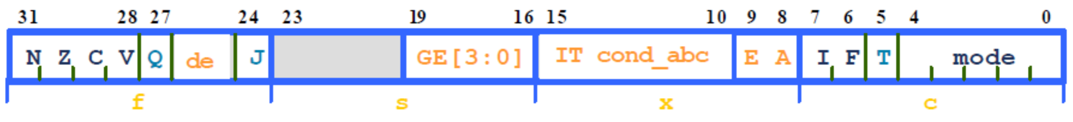


***Figure 4*: ARM Status Register [1]**

Integrated Systems

Generally packages contain a solitary Integrated Circuit (IC) or transistor within. In a functional system there are multiple components, collectively used together to complete a designated purpose. It can be extrapolated from this that a circuit board supporting a system would have several packaged ICs contained on it. Each of these components is then connected where necessary by using traces on different layers. While this is functional and manageable with intelligent layout, it can take up large amounts of space on the board. This can be an unfortunate consequence because many devices are subject to severe space limitations in their design, which becomes difficult with multiple packages on the same board each making its own footprint. This is especially problematic once the number of necessary traces for each package is considered. With the potential of hundreds of leads each, this is particularly true with modern high end microprocessors. Combine that with the need for memory and other system components to fully function, this rapidly becomes an expensive and difficult proposition. Some devices complicate this further, such as mobile handsets, which are constantly and simultaneously becoming sleeker and increasingly overloaded with a user functions that require new parts.

To conserve space, a natural solution is to package some of these commonly paired components together. The three most common design approaches are System-on-a-Chip (SoC), System-in-Package (SiP), and Package-on-Package (PoP) [6]. While small differences between these exist, they are frequently used interchangeably in conversations and in papers. This confusion is understandable due to the end result between all three approaches being the same; a full system is contained to one footprint. Another contributing factor to the misuse of nomenclature is that these advanced packaging techniques are not mutually exclusive; it is possible to have a combination of all three integration techniques. All interconnectivity of each of the components that make up the entire system is handled within the design. For the purposes of this paper all designs that utilize these approaches and their variants will be referred to as integrated systems.

System-on-a-Chip

The first of the space saving strategies, SoC, is the practice of putting several different system components on the same wafer die. Because all components are on the same plane, this is considered a 2-D packaging technique [8]. These types of chips commonly involve the use of different IP designs individually purchased [29] [30]. SoC has the advantage of almost always being the smallest and cheapest solution, and there is no compelling reason to use another method if this will do the job [31]. However, there are situations that are compelling enough to utilize other packaging technologies; to name a few, it adds stress to die size constraints [8] [32] and memory is difficult to include [8] .In fact it is generally considered better practice to use a different packaging technique for memory [6].

One such device is the Texas Instrument's Open Multimedia Application Platform (OMAP), which utilizes application end ARM cores. As the name implies, this particular example of a SoC is an ASIC targeted at media applications. In addition to the ARM RISC core, there is also a TI developed DSP core included, a shared memory system between the two, as well as other system components [9]. Other significant ARM-based application oriented SoCs exist on the market; these include the Samsung Hummingbird [33], Qualcomm Snapdragon [33], Nvidia Tegra [33], and Innovative Multimedia Extension (i.MX) [34].

System-in-Package

In contrast to the SoC approach, SiP places several different dies in the same package, and uses wire bonding between the dies [6] [7] [29]. There is a small amount of ambiguity surrounding the definition of SiP. Some sources [31] take a broad definition by declaring a SiP to be any package with more than a single chip, and then defining a variety of subtypes such as Multichip Modules (MCM) and Multichip Packages (MCP). PoP is frequently included among these subtypes as well. Occasionally, a more specific definition is used, identifying SiP as a 3-D technique consisting of a vertical stack of chips [6] [8].

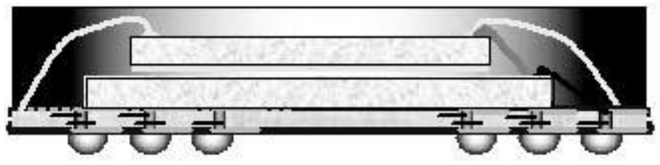


***Figure 5*: System in Package [6]**

This more detailed definition usually accompanies a second separate term for the practice of multiple chips being placed on the same plane and possibly board. This 2-D counterpart is designated as a System-on-Platform (SoP) [8]. For purposes of this paper, SiP will include all techniques that involve a single package containing more than one chip, thus separating PoP from the others. *Figure 5* shows the cross-section of a SiP, with two chips encapsulated in a single package.

SiP has a variety of advantages over a purely SoC approach. This is particularly true when dealing with the subject of memory as noted in the *System-on-a-Chip* section. Also some components are difficult or impractical to place on a SoC [31]. These are examples of viable reasons to use a SiP approach. Despite these advantages, SiP still faces some complexity and cost issues because of the wire bonding challenges between the different chips [29]. It also has higher power consumption [32]. It is clearly demonstrated that both systems have their strengths and weaknesses. Due to this, it is the conclusion of experts that both of these systems will coexist depending on the needs of the solution [31] [32].

Package-on-Package

The final major classification of integration techniques, PoP, is the practice of stacking different encapsulated packages on top of one another. PoP holds the same advantages over a pure SoC solution that SiP does. Though, between the two, other comparisons, aside from the obvious inclusion of extra encapsulations in PoP, can be drawn. It should be noted this technique comes at the cost of a larger footprint [6], though the payoff is considerable. First of note is the improved memory options, SiP requires special and customized memory footprints, whereas PoP is designed to allow standardized footprints, thus any standard memory component is valid and useable [6]. In a similar fashion, almost any ASIC IC holds the same advantage; they can be individually packaged and use a standardized footprint [6]. Not requiring customized interfaces to fit additional dies in the same package makes IC procurement much easier, thus PoP allows cleaner and easier business deals during creation of these systems [6]. Also reliance on wire-bonding methods is heavily reduced with innovations such as through-silicon vias (TSVs) [35], standardization in packages to support PoP [36], and implementation of flip-chip Ball Grid Arrays (BGA) [37].

A cross-section of a PoP design is shown in *Figure 6*. This particular image is actually that of the system included on the Beagle Board discussed in the next section. There are two stacked packages in this image. The bottom package contains a single die, which is actually an OMAP. The top package contains two dies, one for flash memory the other for SDRAM. To conclude, this is a fantastic demonstration of the different integrated system techniques; this is a PoP containing an SoC in the bottom package and a SiP in the top package.

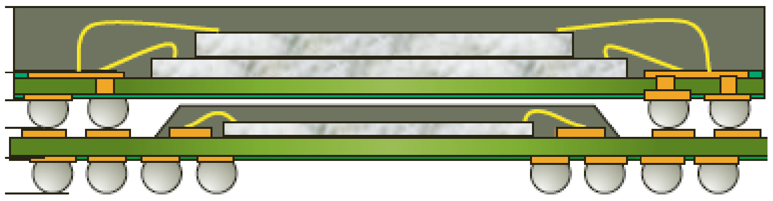


***Figure 6*: Package on Package Cross-section Using BGA Packaging [38]**

BEAGLE BOARD

Open-source software is fairly common, ranging from small applications to full operating systems such as Linux [39]. These programs are familiar to a variety of user communities, and allow for free use of the program as well as unfettered access to the source code. The complete access to all development resources enables user generated modifications and development. Occasionally open-source hardware devices are also released for experimentation. Similarly to software, the schematics, Bill of Materials (BOM), Printable Circuit Board (PCB) layouts, and all other information is released for free [40]. One such device is the TI Beagle Board, which was created specifically to be an open-source hardware product. Though it was aimed at hobbyists [41], the device was developed with the intention of familiarizing development communities, and particularly university students [42], with OMAP driven products.

The Beagle Board, seen in ***Error! Reference source not found.*** below, is designed to allow obbyists to experiment with TI’s OMAP3530 PoP processor. The board has gone through a variety of updates and revisions; the specific version seen in ***Error! Reference source not found.*** s revision C4. Along with each revision, a full user guide is published alongside it that contains all information expected with open-source hardware devices such as the BOM and detailed overviews of each component. It should be noted that the board only offers a minimum set of features and is not intended to be used in end products [38]. It is instead focused towards starting projects and experimentation. The Beagle Board is still equipped with a suite of standard input and output (I/O) interface components, debugging interface components, and has multiple expansion capabilities. The Beagle Board has been used in a variety of projects, and a large community [43] has emerged around it, even supporting annual tournaments.

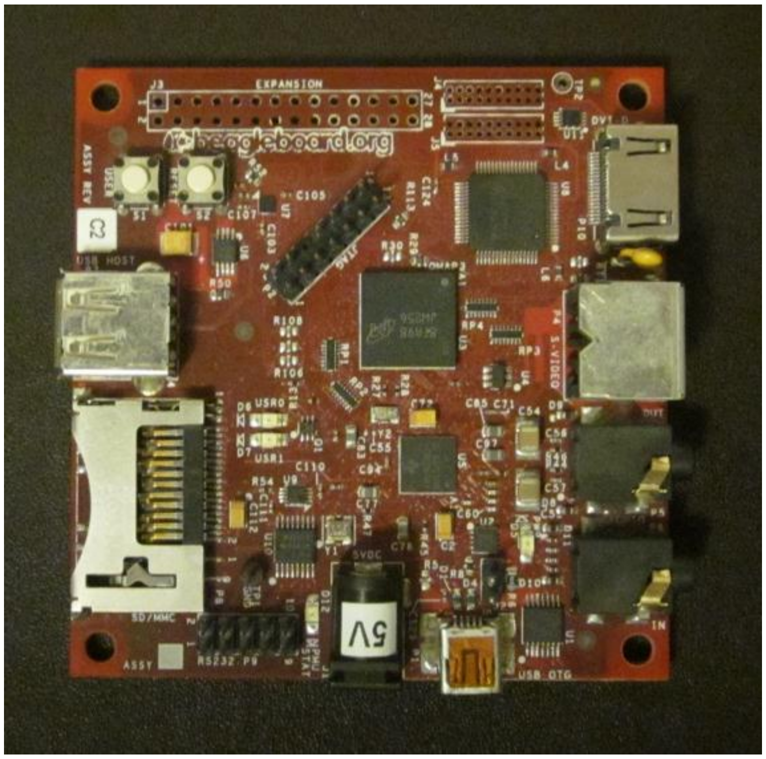


Figure 7: BeagleBoard Rev C4

Specifications

Mechanically, the board was designed to take up minimal space. The Beagle Board was designed on a six layer PCB. It only encompasses an area of 3.0 inches wide by 3.1 inches in length. It should also be noted, the board is designed to allow daughterboard devices to be attached to its underside. From an electrical standpoint, low power was a key consideration. It is able to fully operate on a 5V supply and drawing only 350 mA.

Interface and Extensions

This section examines the specific features of the C4 revision of the Beagle Board. Excluding the expansion board connection, there are thirteen different sources of interface with the Beagle Board. These are detailed in

Table 3 and numerically labeled on *Figure 8*.

The board is designed to function by using the USB On-The-Go (OTG) port for both power and communication. Though for both functions there are alternative options. For power, a jack is located on the board providing the option of using a 5V DC power supply. It should be noted that the USB Host port does not have sufficient power to run most USB devices without use of the power jack [38]. For communication, in addition to the USB inputs, a 10-pin header is included to allow access to the RS232 serial port, though this method is cumbersome and requires several obscure converter cables.

Because the OMAP is a multimedia focused platform, audio and video I/O components are included on the board. The audio uses a simple 3.5mm stereo jack for both input and output. For video there are two different output options. The first is S-Video and second is DVI-D, though the DVI-D out actually uses a HDMI connecter for space conservation, thus requiring a converter cable. There is also an option of connecting a small display or reading data off of the LCD headers.

The remaining interfaces are as follows. There is a JTAG for advanced debugging by use of an emulator. Also, a 6-in-1 MMC/SD device is used for enabling a variety of MMC+ supported devices. There are four status LEDs, three of which are controlled by user software and the final is a power indicator. Finally, two buttons are included on the board. The first is a reset button and the second is labeled the user/boot button. The second button can be used in conjunction with the reset button to change the boot order; alternatively, user software determines its purpose.

Table 3: **OMAP Interfaces and Beagle Board Connectors**

	Interface	Connector
1	USB OTG	USB Mini AB
2	USB Host	USB A
3	Optional Power	5V DC
4	JTAG	14-pin Header
5	Serial (RS232)	10-Pin Header
6	S-Video	S-Video
7	DVI-D	HDMI
8	Stereo Out	3.5mm L + R
9	Stereo In	3.5mm L + R In
10	Indicators	N/A
11	Buttons	N/A
12	SD/MMC	6 in 1 SD/MMC/SDIO
13	LCD Connection	Two 2x10 Headers

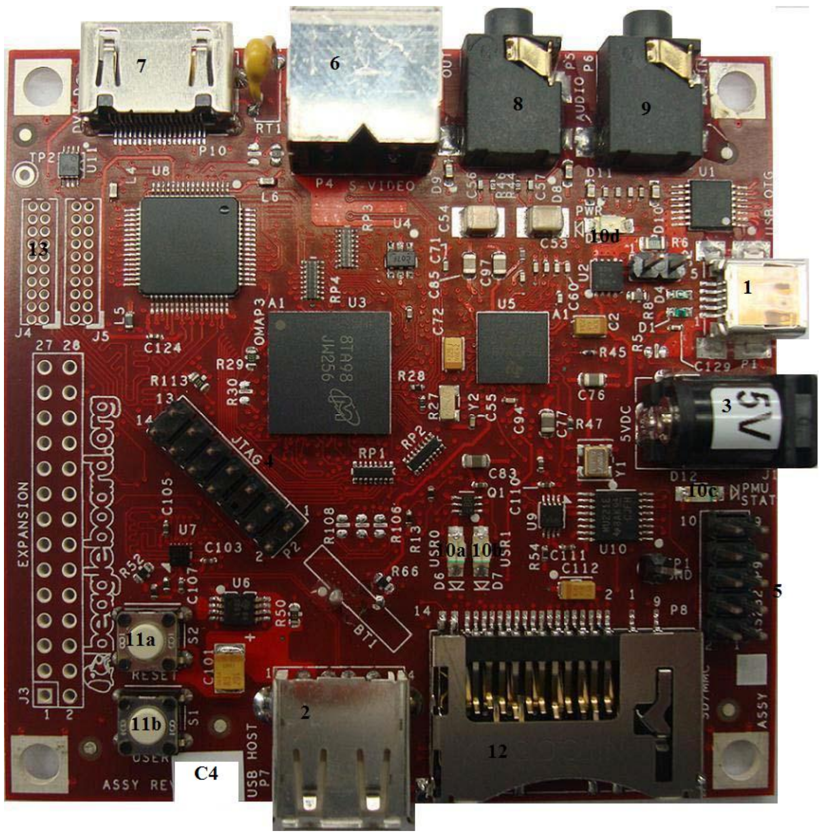


Figure 8: **Interfaces [38]**

There is an expansion socket provided that allows for additional functionality. New boards can be developed to take advantage of this 28 pin header to add more specific capability. A couple examples of these include an OLED display [44] or a lithium ION battery pack [45]. There are also expansions that don't use the socket such as the Flyswatter [46] for the JTAG.

OMAP3530 and POP Memory

The Beagle Board uses a .4mm pitch PoP package with an OMAP3530DCBB72 720MHZ processor on bottom; the top features both NAND and SDRAM [38]. This is the specific configuration observed in the illustration (*Figure 6*) used to demonstrate PoP packaging in the section above.

As with any other series of OMAP processors, the OMAP3530 is a SoC that targets multimedia applications. Utilizing the Cortex-A8 core, the OMAP is fully capable of running several different operating systems. A comprehensive list of specifications is included in

Table 4. To see how the other system components in the OMAP interact with the processor, examine the block diagram provided in *Figure 9*.

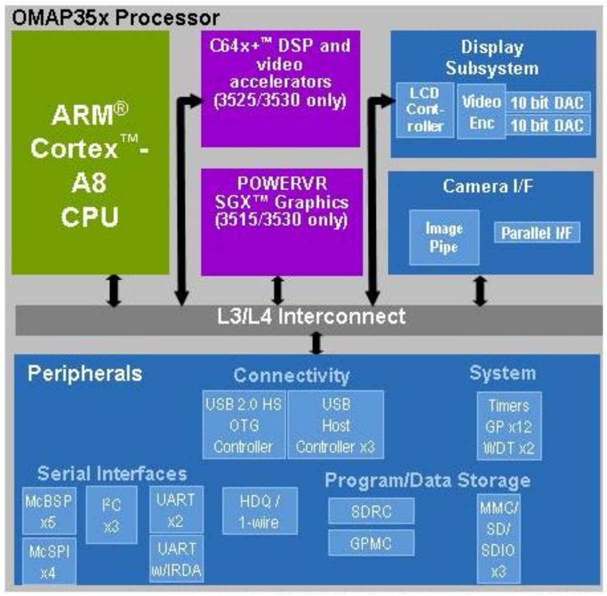


***Figure 9*: OMAP35xx Block Diagram [30]**

***Table 4*: OMAP3530 Parametrics [30]**

	OMAP3530
CPU	1 64x+,ARM Cortex-A8
Peak MMACS	4160
Frequency(MHz)	520
RISC Frequency(MHz)	720
On-Chip L1/SRAM	112 KB (DSP),32 KB (ARM Cortex-A8)
On-Chip L2/SRAM	96 KB (DSP),256 KB (ARM Cortex-A8)
RAM(KB)	64 KB
ROM	16 KB (DSP),32 KB (ARM Cortex-A8)
EMIF	1 32-Bit SDRC,1 16-Bit GPMC
External Memory Type Supported	LPDDR,NOR Flash,NAND flash,OneNAND,Asynch SRAM
DMA(Ch)	64-Ch EDMA,32-Bit Channel SDMA
Video Port (Configurable)	1 Dedicated Output,1 Dedicated Input
Graphics Accelerator	1
MMC/SD	3
McBSP	5
Pin/Package	423FCBGA, 515POP-FCBGA
POP Interface	Yes (CBB)
I2C	3
McSPI	4
HDQ/1-Wire	1
UART(SCI)	3
USB	2
Timers	12 32-Bit GP,2 32-Bit WD
Core Supply (Volts)	0.8 V to 1.35 V
IO Supply(V)	1.8 V,3.0 V (MMC1 Only)
Operating Temperature Range(°C)	0 to 90,-40 to 105

The Cortex-A8 belongs to the applications series of ARM processors. All of the common features described in *ARM* section attributed to the application processors are included, though it is notable that the A8 is specifically a single core design [47]. This particular processor is developed to operate in frequency ranges of 600MHZ to 1 GHZ, and uses an integrated L2 cache [27].

Cache sizes are displayed in the above table. Two pipelines are featured in the Cortex A-8. The main pipeline is superscalar, *13* stages long, and utilizes in-order execution [48] [49] [50] [51]. The NEON unit utilizes a 10-stage pipeline for the SIMD based media instructions [50] [51] [52]. The core's block diagram is included below in *Figure 10*.

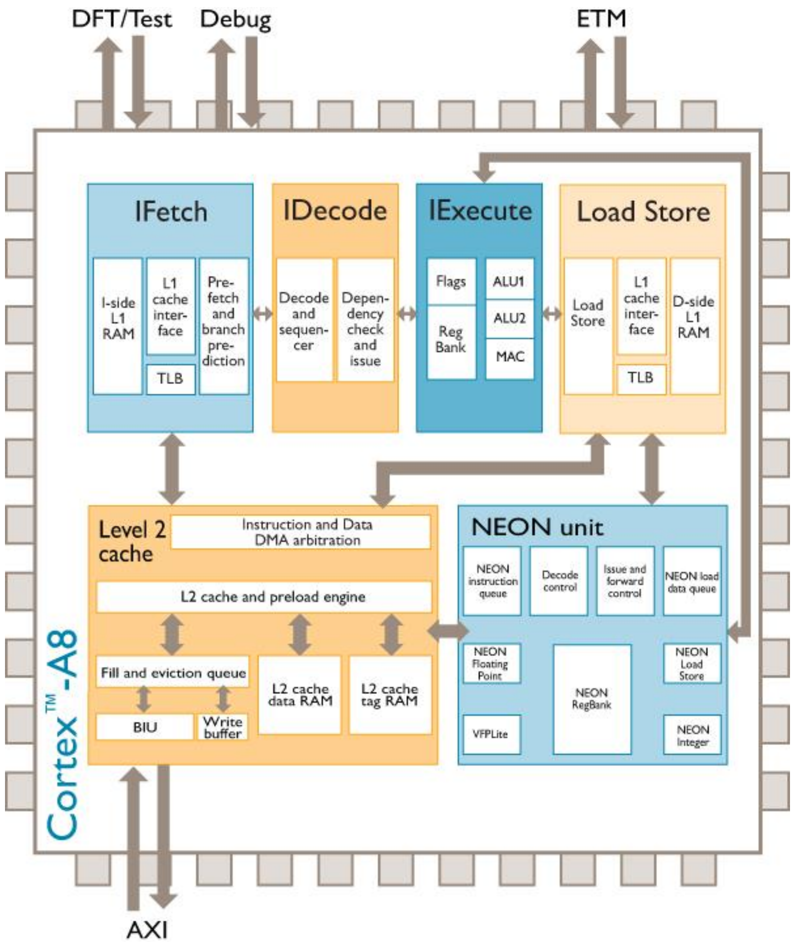


***Figure 10*: Cortex- A8 [47]**

The top-mounted memory used in the revision C4 Beagle Board consists of two different memory components. The first component of memory is the 256 MB of NAND Flash, and it is the default boot device order unless the USER button is pressed. Also included in the PoP memory is 256 MB of DDR SDRAM, which runs at 166 MHZ.

BENCHMARKS

The specifics of benchmarking have been the subject of debate and research for decades. It goes without saying that using the same tools to measure two different systems is the only fair way to compare them. A variety of different benchmarking suites have become industry standards for this reason. It also can be safely concluded that it is important to ensure a thorough and fair application of the suite to each test subject in the comparison to prevent skewed or biased results. However, in practice there is rarely a perfect suite for the job, and misuse of the benchmark suites is frequent [53] [54]. Those are issues with benchmarking that arise in the best of circumstances, however, in the case of integrated systems (as defined in this thesis); the situation is dire. The search for a quality benchmark suite that tests all the functionality of an integrated system, without being designed for a full CPU, leaves much to be desired. Examining what properties make a standard benchmark and how they are misused will be observed in this section, though the results of research to find benchmarks for integrated systems will be explored in *Chapter 4*.

Standard Benchmark Qualities

There are a variety of benchmarks used as industry standards, most of which focus on a specific application or platform. Media, microprocessors, and server towers are among some of these focus targets. The costs and accessibility of these benchmarks are as varied as their purposes. Upper end benchmarks include the Standard Performance Evaluation Corporation (SPEC) [55], EDN Embedded Microprocessor Benchmark Consortium (EEMBC) [56], and Berkley Design Technology, Inc. (BDTI) [57]. The former consists of a large number of different suites, and has been widely popular; it has at times comprised over half of reported conference benchmark results [53]. The latter two examples are both embedded processor oriented suites. On the other end of the spectrum from these proprietary benchmarks, a large number of open-source benchmark suites exist. One worthy of note is Dhrystone [54]. Dhrystone is several decades old, though it is still used today as a popular synthetic benchmark choice for integer operations.

Ideally a benchmark that is treated as the industry standard should have certain properties associated with it. It should be thorough so that it tests the entire system or application. A good benchmark should not be biased; in other words, it should be representative of normal conditions and software used [58]. Additionally to be meaningful, it should be difficult for vendors to design a system that does well in the benchmark without actually being a good system. This is relevant because vendors are known to cheat [53] [54]. Another important quality is that a standard benchmark should be current with modern specifications [54]. Clear guidelines or standards on running and scoring the benchmarks should also be provided so results can be consistently reproduced and relied upon [53] [54]. They provide some metric of measurement so that the results can be fairly compared; generally results should be easy to understand and relate to other metrics. Popular benchmark developers actually certify results (or hire third party businesses) to improve the trustworthiness of their product [59].

Misuse

Even using an ideal benchmark, there are several ways to incorrectly exploit results. Misuse can be defined as employing the benchmark in some other way than intended. This can come in a variety of ways, from ignoring a few guidelines laid out by the benchmark to designing a system in a way to maximize the benchmark results specifically. Whether deliberate or accidental, benchmark misuse skews results, sometimes considerable amounts [53], which can falsely advertise the tested product or mislead future research.

One major source of misuse is the failure to follow the guidelines of implementing the benchmark. Incompletion is a good example of this. One paper extensively examined the different ways SPEC was misused [53], and one recurring theme was incompletion. It found that it was common to not run all the programs in the suite, and that less than a third of research papers even provided a reason why. Those that did stated they were only examining the expected areas of increased performance, or couldn’t get all the programs to run. Both of these answers should be taken as a red herring to the results. Speedup results in SPEC are calculated based on using all the programs, so this leads to a misuse in the scoring as well, because assumptions had to be made for the missing programs. A similar fallacy existed in not running a program in the suite to completion, and then extrapolating the results from a sample from the beginning. It doesn’t take much contemplation to see the danger in that approach. Though the paper does make a point to show that these can all be understandable in certain circumstances, it is still a misuse of the intent of the benchmark.

Age is also a consideration for benchmark misuse. As new systems are developed and change, the validity of a program begins to decrease. This is particularly true with Dhrystone [54]. The white paper on Dhrystone indicates that the benchmark easily fits in most modern L1 caches, meaning it is worthless for testing memory stress. Despite this, people still use Dhrystone. The same thing was noticed with the study on SPEC misuse; though SPEC95 had been discontinued, a large number of research papers were still using it [53]. Even if these benchmarks are correctly used, it just isn’t reasonable to use them on modern systems as most of the tests are no longer valid or thorough.

The above prevalent examples of misuse are generally innocent, or at the very least understandable. Unfortunately it is not unheard of for a more intentional and debatably malicious form of misuse to occur. Using favorable assumptions to oversell the results is one example of a twist on the above. Though even worse than that, designers are very capable of making a system in such a way that it ‘tricks’ the benchmark by optimizing their system to specifically score well on an industry standard benchmark. Dhrystone is particularly infamous for this because of how easy it is to do [54].

Prolific Benchmarks

As previously defined, one of the most popular standardized benchmarks for embedded systems is BDTI [57] [60]. Primarily, their benchmarks are targeted towards signal processing capabilities, though they have some less used application benchmarks. BDTI is considered a respectable standard, and offers a great amount of reliability and trustworthiness to its clients. They even post summarized results of results for specific cores that have been tested with BDTI benchmark suites. The Cortex-A8 is among these and can be located on their website [61].

The other significant powerhouse in the embedded systems benchmark market is EEMBC [60] [62]. This consortium provides a multitude of benchmark suites that cover a wide array of targets based on application focus. Much like BDTI, they do publish their results online, though it is much more detailed. The specifics of certification are also well guarded, another common ground with BDTI [63]. BDTI does claim technical superiority in the rigorousness of their benchmarks [63], particularly in the DSP market. Another downside that has been noted is that some of the specific tests within the benchmark suites are not well-thought out or representative of realistic conditions [60]. EEMBC operates in two modes [63]. First is an out-of-the-box mode that uses non optimized code and is noted as truly fair, though not realistic. Alternatively, the option to optimize code (C or assembly level) is available, though there are no guidelines or recommended approaches for it which makes it difficult to fairly compare the results. These flaws and complaints do not outweigh the benefits of EEMBC, there are plenty of valid reasons they are accepted as a standard.

On the other end of the spectrum there is SPEC [55], a benchmark suite for high-performance computers. It also happens to be one of the most popular industry standard benchmarks [53]. SPEC tends to utilize neutral programming language to improve its diversity, and also features many different suites with specific design goals. SPEC also provides rules on implementation, which makes comparisons that use this benchmark correctly trustworthy.

Another rather popular benchmark is Dhrystone [54] [63]. This is an integer synthetic benchmark that is still used in many embedded systems despite its age. ARM Holding uses it to help advertise their processors on their product pages [27]. Its popularity can likely be attributed to being open-source as well as the widespread use generated from being the first to successfully use a single score as a performance indicator [64] .

One open-source benchmark suite aimed at embedded applications is Mediabench [65]. The suite consists of media and telecommunication applications. Unfortunately, the original suite requires software that has been discontinued, or is difficult to find. There is a sequel suite, Mediabench II, that has seen some development but seems largely incomplete and abandoned. Although these benchmarks are not available, they were very popular at one time. During research and literature review, this particular benchmark suite came up in some recent benchmark surveys [58]. It was worth mentioning due to the potential a completed version might have had as well as the frequency it was encountered.

III. IMPLEMENTATION

In preparation for running any benchmarks or other tests, it is necessary to choose a platform to run them. The Beagle Board is ideal for this purpose. To prepare the beagle, it must be set up to run similar to a personal computer complete with an operating system and hardware peripherals. Ubuntu [66], a Linux distribution, is selected as the OS. Equally necessary is an appropriate benchmark suite. MiBench [10], the benchmark used for this experiment, is non-application specific and focuses on generic embedded systems use.

The Beagle Board suits the purpose of benchmarking an integrated system for several reasons. First, the integrated system on the Beagle Board exceeds the basic criteria to be considered an integrated system; it is an exemplary example of such. Secondly, by nature of its design it is easily accessible for experimentation. In fact, as previously explored, that is the intent of the Beagle Board. Finally, the board has a large following and support community providing more readily available software resources and user guides.

Although a myriad of benchmark suites do exist, MiBench suits the purposes of this experiment. Some of the proprietary benchmarks were discarded for sheer cost reasons (BDTI and EEMBC), and others were legacy (Media Bench [65]). Hardware constraints also played a part in selection. Because MiBench is general purpose, many appropriate test areas are covered.

Preparing MiBench to run on the Beagle Board is not the only necessary task in order to prove or disprove the thesis; in reality it is far from it. However, the process of attempting to find the most applicable resources for the experiment does provide a thorough experience to help gauge availability of benchmarks suited to the needs of ARM-based application integrated systems. Other research focuses are mandatory. It is imperative to survey some of the overall better suited suites and research results of past benchmark tests on germane systems by other parties as well.

BEAGLE BOARD SETUP

Preparing the Beagle to run benchmarks is a muti-step process. To do so requires obtaining compatible hardware peripherals. Most of them were common and easy to acquire, but others were more arcane or very specific. The standard I/O used can be inferred from

Table 3 listed in the *Beagle Board* section of *Chapter 2*. This does require a USB hub, however, for full usability. A special crossover cable (IDC10-DB9) working in tandem with a null modem cable is needed for serial communication with the device. The second step of set up is to get an operating system to functionally run on the Beagle Board. There has been lots of effort in making the different significant operating systems available in the past. Though the best supported and easiest to implement for this are Linux distributions. While there have been many projects to bring the various Linux distributions to the Beagle Board, Ubuntu is best suited to the task. Also it has the most community support among the Linux distributions. However setting up Ubuntu to run on an SD card is still an arduous task.

Setting up Ubuntu for the Beagle Board

The support sites for the Beagle Board make the ARM binary interface of the most recent versions of Ubuntu available. During the time of this exercise, the most recent stable version of Ubuntu is 10.10 (Maverick Meerkat). The approach used to run Ubuntu requires use of a SD card for the kernel and root file system. The card used for this was a 16GB SDHC Kingston device. The first necessary step is to format the card into two specific partitions with specific geometry; the first is the boot partition (FAT32), and the second (ext2) is for the root file system. The boot partition contains a pair of beagle board specific boot loaders and the kernel image. Older versions of Ubuntu were not available and users had to make their own image and root file system copies using recommended software. For more details on partitioning the card see Appendix A. Secondly, if using an older version of the Beagle Board it is necessary to update the x-loader on the NAND flash to get the latest versions of Ubuntu running. This requires a serial connection and a copy of x-loader placed on a SD card. One can manually overwrite the previous version in this way. Next it is possible to boot from the card, though the initial boot does take a considerable amount of time. Unfortunately the images that are available are very limited in features, and do not even include a GUI. Thus the fourth and final step is the simple but time consuming process of acquiring and installing enough applications for a comfortable working environment. One nice feature of using the SD cards with the boot loaders established on the first partition is that one can easily interchange different cards with different operating systems.

MIBENCH

MiBench [10] (pronounced “my bench”) is an open-source benchmark suite designed specifically for embedded systems. This suite was developed in 2001 at the University of Michigan – Ann Arbor. When MiBench was developed ARM was still emerging, so they designed it to be compatible for many differing ISAs by using C source code for all benchmarks. At the time there was not a clear dominating ISA; nothing as powerful or comprehensive as something like the Cortex-A series existed. To their credit they did include tests for floating point units (FPUs), even though that was not a common feature yet. Other embedded systems benchmarks from the same time were much more single application focused. Despite legacy design goals, many of the tests in this suite are still relevant in purpose. One area it is very weak on is media applications. As an area that ARM has made large strides in, this makes it an incomplete benchmark for beagle board. However, on the other side, media specific benchmarks usually don't focus on many of the other areas that this one does have.

Composition

Taking the stance that the embedded system domain has a wide range of applications, MiBench, in turn, attempts to provide a wide range of benchmarks. These are broken into six primary categories based on the most common embedded system applications: auto/industrial, consumer, office, network, security, and telecommunications. MiBench has a set of *35* embedded applications across these categories. Many of the benchmark tests include a short and a long version within them. *Table 5* shows a summary of each category set.

***Table 5*: MiBench Categories**

Auto/ Industrial	Consumer	Office	Network	Security	Telecomm
basicmath	jpeg	ghostscript	dijkstra	blowfish enc.	CRC32
bitcount	lame	ispell	patricia	blowfish dec.	FFT
qsort	mad	rsynth	(CRC32)	pgp sign	IFFT
susan(edge)	tiff2bw	sphinx	(sha)	pgp verify	ADPCM enc.
susan(corner)	tiff2rgba	stringsearch	(blowflow)	rijindael enc.	ADPCM dec.
susan(smoothing)	tiffdither			rijindael dec.	GSM enc.
	tiffmedian			sha	GSM dec.

The first set of tests, automotive and industrial control, is somewhat self-explanatory; it focuses on applications that are found in control systems. These focus on basic math functions, sorting, bit counting, and shape recognition. The next set, consumer, is aimed at the market of consumer devices, and is therefore the most applicable set to the Cortex A-8. They include image compression and MP3 encoding and decoding to name a few. This set is lacking in video and other modern multimedia expectations, and indicated prior. The third set focuses on embedded processors primarily found in office appliances. Thus it is primarily deals with text specific programs. Fourth, the network set is focused on applications dealing with the kind of application found in networking devices. Shortest path algorithms and tree lookups are prime examples of the programs found within. Security benchmarks include programs that run hash algorithms and encryptions. Finally, the sixth category is telecommunications. Included within this are tests specific to frequency analysis, checksums, and voice encoding/decoding. For a specific detail of each benchmark, check Appendix B: MiBench Details.

Execution

Each benchmark has to be individually installed or compiled with GCC. The MiBench developers include an executable file in each of the benchmarks to be run once ready. These provide a strong representation of the workload associated with the program. Frequently the executables are accompanied by input examples. To differentiate between the large and small versions of each benchmark test, sometimes two separate executables are included, while at other times, different input samples are used.

In preparing to run the benchmarks, simple scripts are written that utilize each benchmark executable five times and store each individual run time to the same file. The script is written in such a way as to go through this process with every large and small version in an entire category, thus there are six different scripts. In addition to the shell scripts, one simple C program is written to provide a precise execution time for each of the benchmarks. The program will accept the benchmark execution command line as a string and complete after the time calculation. This script and C combination is implemented for multiple reasons: reduction of human error, automated compilation of results, providing a sample pool to help isolate anomalous run times, and to make the process less laborious.

Unfortunately, one drawback of MiBench is that it isn't prolific; therefore, it is difficult to relate the results. This is especially true without output metrics, or clear guidelines on how to measure the benchmarks. To provide context, contrast, and scalability the benchmark is also to be run on a fairly standard laptop (Specs in *Chapter 4*). The laptop is in a different category than the OMAP. The intent is not to compare the two; it is only to provide context with something that has been more universally tested. Because the AMD processor in the laptop has more readily accessible performance results, one can easily look them up to provide frame of reference for the MiBench results.

RESEARCH FOCUSES

Two primary research focuses can help answer some of the questions that were asked in the thesis introduction regarding benchmark availability for integrated systems. The first of these research focuses pertain to finding any available benchmarks that are in use as well as being relevant to the target platform. Within that goal, it would be appropriate to ensure that using them does not automatically generate benchmark misuse to make them apply to meet these needs. Similarly, it is also necessary to make sure the benchmarks are comprehensive enough. The second topic of study is oriented around finding the results of past benchmark results of integrated systems comparisons. This focus can be realized by seeking literature that has already compared different application integrated systems. Another avenue to pursue within this focus is searching the results databases stored by some of the larger benchmark suites. Whether or not an existing suite has been effectively used in the aforementioned task, and if not, what methods have been used for providing comparison will be discovered in this way. This second focus will also serve to indicate both where applications processors have been tested against full processors or microprocessors and to which classification they better belong.

Benchmark Criteria

Due to the hybrid features of applications processors, particularly when considering the already ambiguous nature of distinguishing processors and microprocessors, it is necessary to survey benchmarks from both classifications. Aspects from benchmark suites of both of these classifications will be found to be relevant. However, the search is for a suite that successfully manages to test both ends of the applications processors. More than that, to meet all the sought after criteria of being applicable to integrated systems, the suite must be capable of testing a full system instead of just the core.

The benchmarks that are found to be acceptable will have to feature several specific qualities. The most obvious of which is that it is comprehensive enough to not require secondary benchmarks to fill in untested capabilities. In contrast, an overly generic benchmark would fail to thoroughly test the system’s purpose and capabilities. It should be a forgone conclusion that the benchmark should have all the qualities (or be able to produce them) expected of a standard while not being susceptible to intentional misuse. These are just a few of the more significant pitfalls that might make a particular benchmark ill-suited for the task. Other benchmark specific issues may also disqualify them from being suggested or nominated to be the standard.

IV. RESULTS

The intent of the research in this thesis is to find the state of availability of benchmark suites appropriate for ARM-based applications integrated systems. As mentioned, the primary focuses of research includes potential suitable benchmarks and what results exist from past comparisons and tests. Because no clear standard or potential candidate is readily apparent, an assessment of the future of the target platform was also included.

Literature is reviewed to find the future of ARM Holding Ltd. and its ISA. Finding the specifications of the IP cores of ARM processors was quite easy. Great detail is made available with technical manuals published by ARM Holding Ltd. Likewise, the literature shows great detail in the expanding techniques to create integrated systems as well as the increasing number of licensees that combine the technologies. Finally, literature indicating the prevalence of applications processors completes the research on the future of the target platforms.

The results of MiBench on the OMAP3530 are included alongside the scale comparison provided with the standard laptop. These are broken into each of the six primary categories outlined by the MiBench developers. Again it should be noted that MiBench is not being proposed as a standard, nor is comparing OMAP3530 against an AMD processor the focus of study. Instead this is merely a sample benchmark along with a medium of comparison.

BENCHMARK RESEARCH

There are many benchmarks and corresponding published results that have been done for ARM cores as well as some or fewer for integrated systems that include them. To underscore the availability of these results, ARM Holding Ltd. posts Dhrystone results on the processor profile pages. Likewise, the two most prevalently used embedded benchmark suites, BDTI and EEMBC [60], post summarized results online that are free to view [61]. Respectable measures for ARM cores such as the Cortex-A8 are widely available. To a lesser degree, finding comparisons [33] and successful benchmark results [67] [68] of integrated systems, particularly SoCs, is also a relatively manageable endeavor.

Fully applicable benchmarks, or results, that met the criteria of being acceptable at a standard quality without being misused (by the definition explored in *Chapter 2*) are not found to be universally applied. In fact, nothing that is found presented itself as an obvious benchmarking standard for ARM-based application integrated systems. Though nothing presented itself as an obvious choice, plenty of material does exist that is worthy of closer examination. It would be erroneous to claim that there are no benchmark suites that merit consideration. Some of the past benchmarks and subsequent comparisons between the different ARM-based integrated systems also contribute to the topic.

Possible Benchmarks

The background chapter provided several prolific benchmarks that are either prevalent in industry or less-known but better apt for being used for measuring the target devices. This sub-section will survey a few of the benchmarks mentioned in this paper (excluding MiBench, as it has been thoroughly explored in *Chapter 3*). The strengths and weaknesses of each of these shall be thoroughly analyzed with respect to the target platform of ARM-based integrates systems.

Despite their reputation for quality and respectability, BDTI does not make their source or certification methods available; thus, it is difficult to interpret at times though it also more difficult to cheat [60] [63]. Their primary benchmarks are too application specific (telecommunications and video encoding) and only measure single-core performance with results that are only relative to one another [60]. They do have a more sophisticated benchmark also used for decoding that can handle multi-core, though it still lacks the diversity needed. To summarize, BDTI meets the quality standards, but is too specific for an applications processor, and offers little to test a full system.

The second benchmark for embedded systems was EEMBC. As mentioned, their results are posted online and certified; one company, Synchromesh [69], dedicates significant resources to certifying and verifying benchmark results [59] [70] to put on EEMBCs database. In addition to services like that, EEMBC is a consortium of major companies in the industry; these attributes make it a very trusted source. Between the different benchmark suites, the EEMBC database does include many products from ARM, as well as many i.MX SoCs in particular. One of their suites, Coremark [64] [71], is free and attempts to provide a single measurement score. Results of several of the SoCs that have been mentioned in this paper can be found in its database. EEMBC clearly meets the needs in quality and widespread use as well as any option could, and it does seem capable of testing the abilities of an integrated system. Unfortunately, it still requires multiple suites to test all the capabilities, and there are few tests outside of Coremark that have been performed for systems other than i.MX.

SPEC was the most prolific of CPU oriented benchmarks. Unfortunately, SPEC does not indicate that there has been any application towards any kind of embedded system, even application processors, despite such a processor’s abilities to run full operating systems. One would think there had been some work done in this area by SPEC, especially with product reviews likening the Cortex-A15 more to a normal processor than embedded systems [72]. However, that is simply not the case. The suite itself is very comprehensive and conclusive, but would still be too difficult for the applications processors, and is certainly not geared towards them.

Stepping away from the mainstream options bring a few new possibilities, such as Dhrystone. Unfortunately, Dhrystone has many shortcomings such as being too small and old to be relevant on modern cache sizes and multicore devices [63]. Another huge problem is the ease in which it can be manipulated [54]. These issues, and others, have been previously mentioned in the *Benchmark* section of *Chapter 2*. Thus, even if Dhrystone might be useable for some embedded systems, the Cortex-A series clearly needs a different representation because it has multi-core capability, large caches, and FPUs. Dhrystone fails to be appropriate even before considering system components.

There are a variety of other benchmarks of varying complexities that might make one wonder why they didn't get considered. Surveying all the available benchmarks that might be merely implementable on the target platform would be a topic unto itself. As more options were explored, they became increasingly specific-purpose driven, single-score oriented, obscure, or a combination thereof. This is not to imply that there are not benchmarks that can be used to some extent. In fact, several single-score specific-purpose benchmarks have been used to compare SoCs; one of these looked at ARM SoCs included in recent cell phone models [67]. Another project ran several suites on older application ARM systems that used the same core [59]. None of the benchmarks included in either test were enough to measure the entire system on their own. The important information to take away from this is that there are at least some benchmarks that can be found that are applicable to ARM-based application integrated systems. It is just also equally important to understand that they aren't comprehensive or detailed enough to meet the criteria of fully benchmarking an entire application integrated system.

Existing Benchmark Results

The previous sub-section explores some of the significant benchmarks that were most likely to apply to the target; it also makes several references to existing comparisons and benchmark results that are relevant. This sub-section intends to explore some of those referenced results to get a grasp on what level of previous work has been done on the topic. This research can help indicate if an available benchmark has been used thoroughly enough to satisfy the objective of fully comparing multiple target platforms. This also serves to underscore how much difference the integrated system components make, which can be used to conclude whether or not only comparing the differences between each ARM core is enough to make a conclusive decision.

The first set of published results to be examined comes from Synchromesh selecting competing SoCs and comparing them using several different benchmarks [59]. The benchmarks used consist of STREAM, BYTEmarks, HINT, and an MPEG-4 decoded/encoder developed specifically by Synchromesh. The tested platforms consist of the i.MX31, OMAP2420, and the Intel Bulverde. The first two both use very similar ARM-11 architectures for their core, whereas the Bulverde uses an older version of ARM ISA. Due to vastly differing base clock frequencies, the i.MX31 is represented twice, once at normal speed and once at half speed. All tested platforms are given as similar operating configurations as possible. Even still, the processors operate at different clock speeds. The results of each benchmark are included twice; once with just the raw results, and once normalized to mitigate the differing clock speed factor.

Stream is synthetic, and is used to measure sustained memory bandwidth in MB/s. By using Stream, it can be noted that the high speed i.MX is superior in performance by its metrics, but the most efficient is the lower clock speed i.MX while OMAP and the higher speed i.MX are similar. To demonstrate this *Figure 11* shows the results of Stream while *Figure 12* clearly shows the same results adjusted for the operating frequency of the platform.

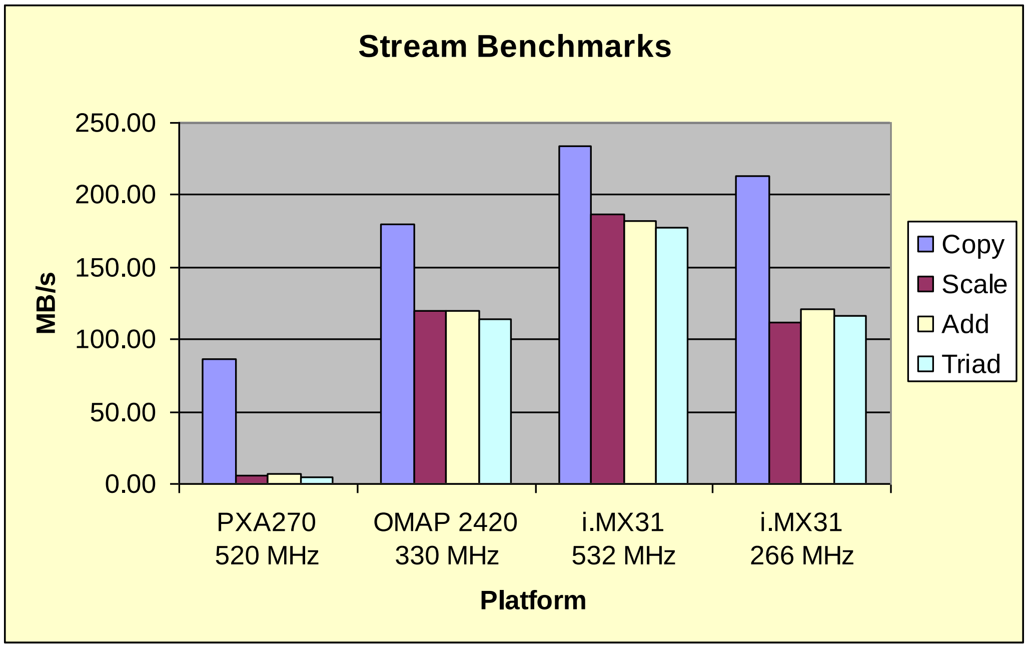


***Figure 11*: Stream [59]**

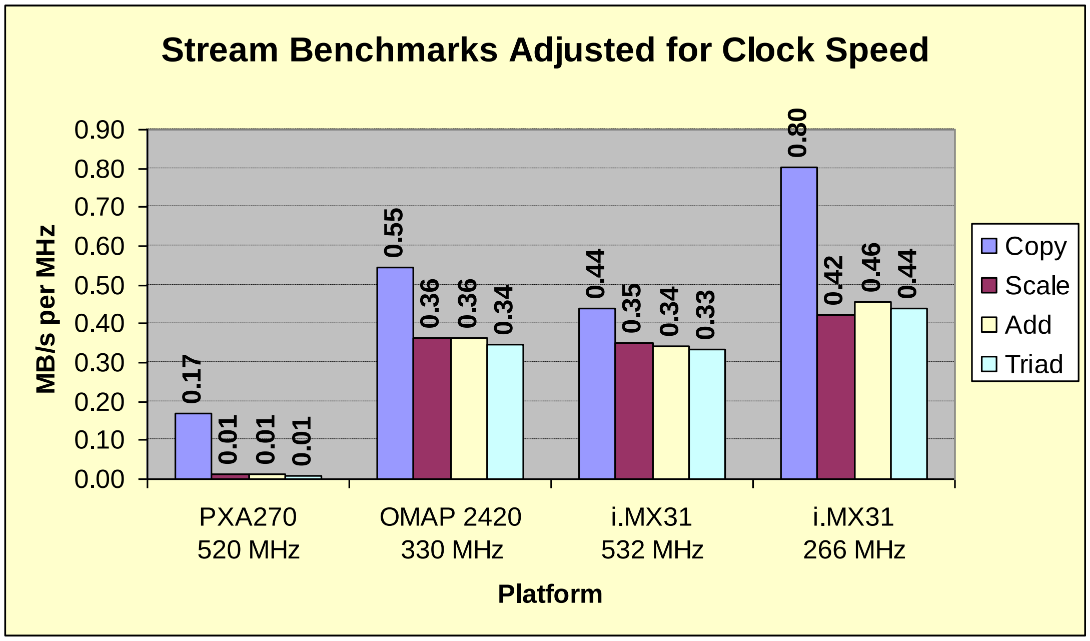


***Figure 12*: Stream Adjusted [59]**

Using normalized BYTEmark results reveal similar trends. *Figure 13* indicated the i.MX scales very well with an adjusted clock speed; the i.MX manages to outperform the OMAP in terms of efficiency despite the operating frequency. It should also be noted from the experiment that the higher speed i.MX outperformed the OMAP.

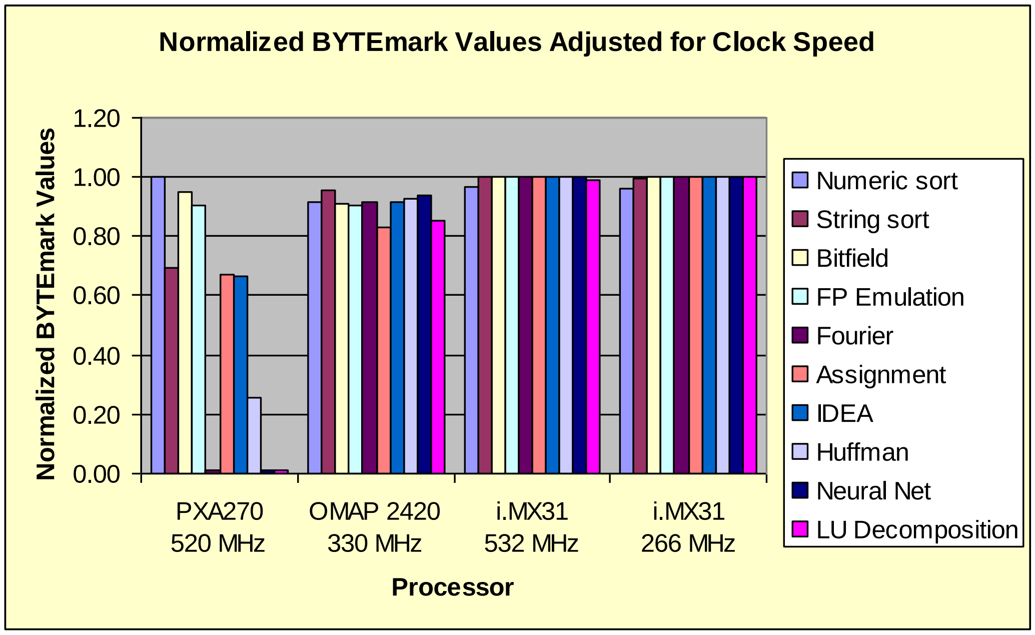


***Figure 13*: Normalized BYTEmark Results [59]**

Synchromesh's comparisons of the OMAP and i.MX continued to indicate similar outcomes for the other benchmarks employed as well. The low frequency i.MX device was the most efficient due to its clock speed being closest to its memory subsystem speed. The i.MX was also concluded to be the highest performing processor at the high frequency clock speed. In some cases even the low speed i.MX outperformed the OMAP. In most instances the processors using the more advanced ARM11 dominated the Bulverde. The results clearly show the memory subsystems included in the i.MX31 make it superior to the OMAP2420 despite having the same core.

A considerably more recent comparison of ARM-based application SoCs that are included in smartphones will be explored below [67]. This series of tests included the single-score benchmarks SunSpider Javascript Benchmark .9 [73], Rightware BrowserMark [74], and GLBenchmark [75]. The primary competing targets utilize the OMAP4430, the Exynos4210, and the Tegra 2. Beyond these, other smartphones using older SoCs were incorporated into the test pool.

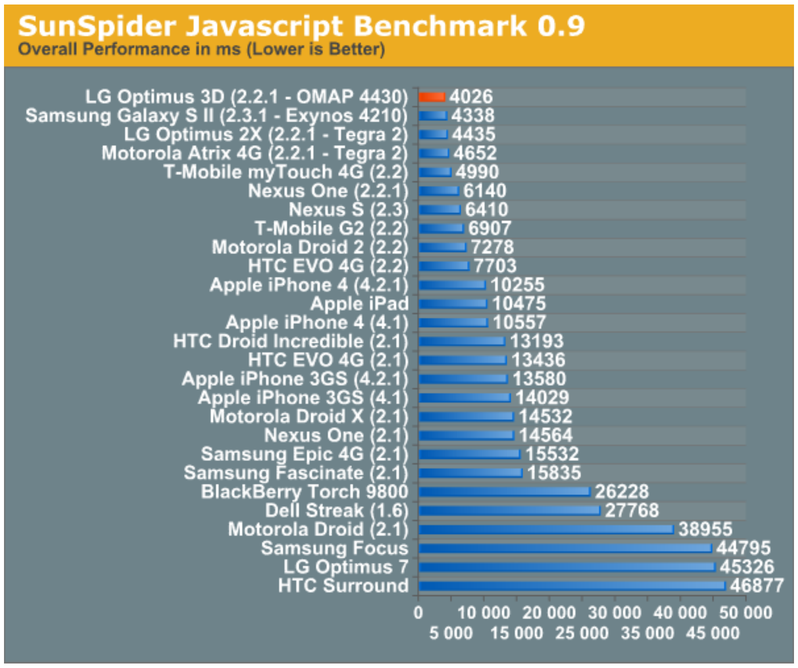


***Figure 14*: SunSpider Javascript [67]**

Though little explanation was included, these main focus devices are all ARM-based applications integrated systems. The OMAP4430 came out the clear winner of this assembly of benchmarks. Looking at *Figure 14* will show that it barely edges out competition in quickness to complete the SunSpider Javaacript benchmark, with the Exynos4210 and two devices using Tegra 2 taking the next three places. It also wins out in the GLBenchmark 2.0 – Pro, a 3D rendering benchmark, as indicated by *Figure 15(b)*. Though, *Figure 15(a)* demonstrates that BrowserMark slightly favors the Tegra 2. This is the only test the OMAP did not come out superior.

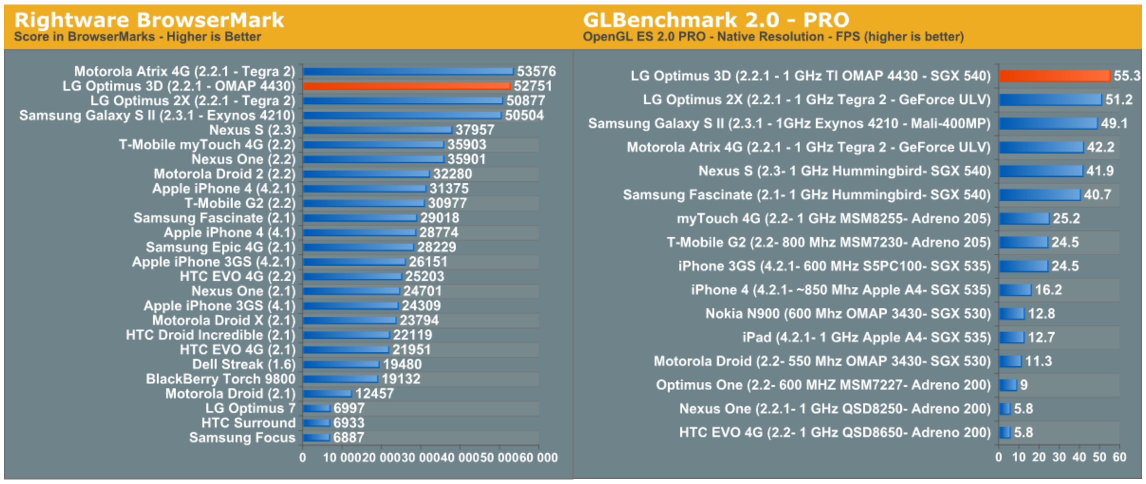


***Figure 15*: (a) BrowserMark (b) GLBenchmark 2.0 [67]**

BTDI, EEMBC, and EEMBC's Coremark also have results that can be looked up online; unfortunately, the specific details of those cannot be repeated here due to copyrights. However, they are easily accessible on company websites, though they do not alter any of implications of the results found here. It can be said that outside of Coremark, most of the results posted are novelties and exist only for a few of many SoC designs. In case of Coremark, only the single-score and test conditions are freely available. As discussed, this would not be useful to thoroughly differentiate between integrated systems. BTDI, on the other hand, did manage to provide a very thorough examination of the Cortex-A8 [61], though they have done little else with ARM much less OMAP or any of its competing systems.

Overviewing existing benchmarking results for ARM-based applications integrated systems highlights several interesting points. First, none of them used benchmarks developed by any of the big standard groups. Second, all testers used multiple benchmarks that were unrelated to one another to make their comparisons. This indicated that not only were single conclusive options not able to be found during the research into this thesis, neither were the experimenting parties able to find them. Third, two different SoCs that had nearly identical cores performed very differently under the same operating parameters. To summarize, no apparent tests nor databases of benchmark results provide a truly comprehensive comparison of ARM-based application integrated systems.

MIBENCH RESULTS

Just as explained during the *MiBench* section under *Chapter 3*, each benchmark was to be run five times to reduce anomalies and provide an average. In order to automate the process, scripts that called a small c program were written. Most of the benchmark suite was straight forward in implementation, with only a handful of unusable benchmarks. In total, all but six of the individual benchmarks were able to compile correctly. The technical reasons for each of these errors were vastly different. Though most of the errors derived from the same common theme, this benchmark suite is just too outdated. Fortunately, every successfully compiled benchmark was successfully run and measured save three exceptions. Most of them had little variance between the five passes indicating quality in terms of precision.

The remainder of this section will be organized by benchmark category and used to explore the results for both of the machines.

Table 6 compares the two different test platforms. Because the benchmark suite lacks any suggested measurement, the best metric is using runtime. The results in the subsections below use the standard of measuring the clock directly before running the benchmark and again immediately afterwards, then taking the difference. For simplicity the monotonic time clock was used for this purpose. The function used is provided from the standard time library for C. It should be noted that this clock does include everything being processed, much like the real time clock, thus cannot be used to accurately determine measurements such as IPC. The original implementation of the benchmark [10], had very precise measurements included such as IPC, branches missed, along with a few others. Unfortunately, these results were obtained by simulation using SimpleScalar, which doesn't help in a real world test.

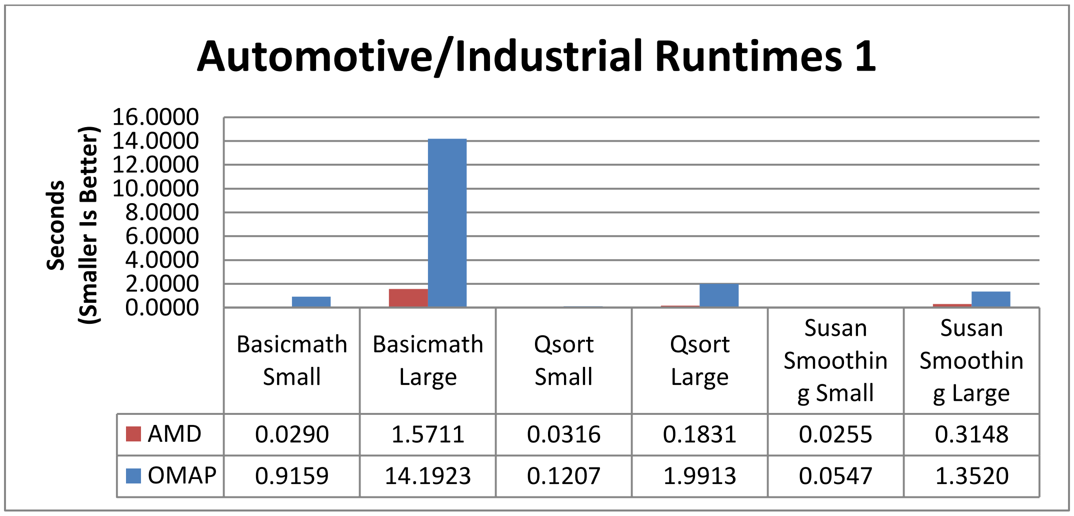
Table 6: **TI OMAP3530DCBB72 [30] vs. AMD Turion ^{TM} 64 X2 Mobile Dual-Core [76]**

	OMAP	Turion
Architecture	ARMv7A	AMD64
Clock Frequency	720 MHZ	2.2GHZ
Pipeline	13 Stage, with separate 10 State Media	Unavailable
Order	Dual Issue, In-Order	Unavailable
L1 Data Cache	16 Kbyte 4-way Associative	64-Kbyte 2-way Associative
L1 Instruction Cache	16 Kbyte 4-way Associative	64-Kbyte 2-way Associative
L2 Cache	256 Kbyte	512 Kbyte
Branch Predictor	Dynamic Branch Prediction with BranchTarget Address Cache, Global HistoryBuffer, and 8-Entry Return Stack	Dynamic Branch Prediction
ROM	2Gb NAND Flash x 16 (256MB) - Expanded with HDSC 16 GB	160 GB SATA Hard Disk
RAM	2Gb MDDR SDRAM x32 (256MB @ 166MHz)	2 GB, 667 MHZ,SDRAM
Operating System	Ubuntu 10.10	Ubuntu 10.10 in VMWare 3.1.3 on Windows 7

Automotive/Industrial

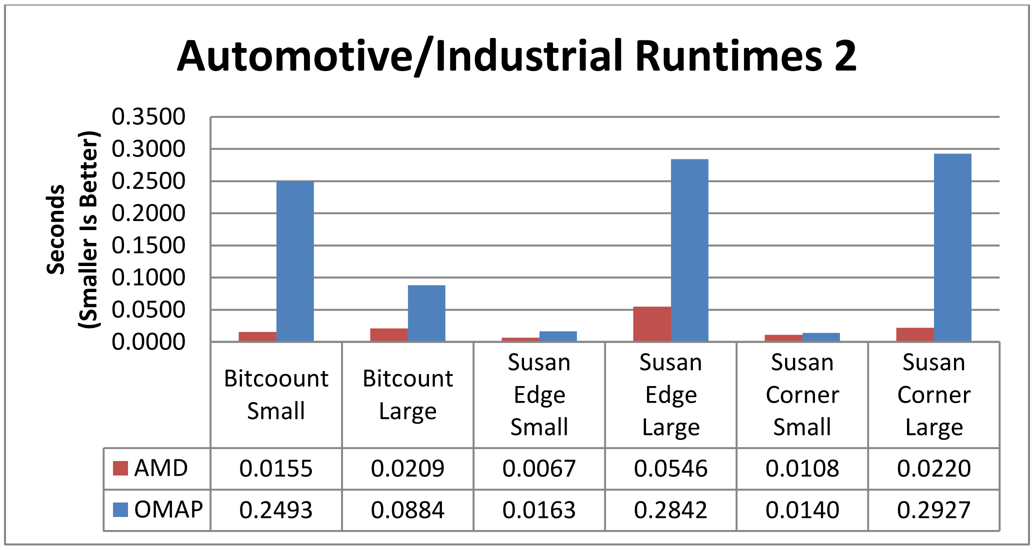
All of the automotive benchmarks were completely able to compile on both machines. Furthermore, they were able to do so without any need of alterations or fixes. This was the easiest test to compile and run because of that lack of complications. Basicmath and qsort output large text files detailing thousands of iterations of the same test, one per line. Bitcount also outputs a text file, though it has a short list of summarized results of different methods. Finally Susan outputs image files with the edges/corners found as well as a final smoothed product.

The timing results from the automotive benchmarks can be seen in *Figure 16* and *Figure 17*. The numbers listed below the bar graphs indicate the average of the five run times, excluding any anomalies. The beagle board in particular has several anomalies in the small basic math and bitcount tests, with one case that took several magnitudes of time longer than the other runtimes. In contrast, the large versions of each benchmarks had very consistent results. Repeated sets of five executions returned similar results, including the anomalies.



	Basicmath Small	Basicmath Large	Qsort Small	Qsort Large	Susan Smoothin g Small	Susan Smoothin g Large
🟥 AMD	0.0290	1.5711	0.0316	0.1831	0.0255	0.3148
🟦 OMAP	0.9159	14.1923	0.1207	1.9913	0.0547	1.3520

***Figure 16*: Automotive Runtimes 1**

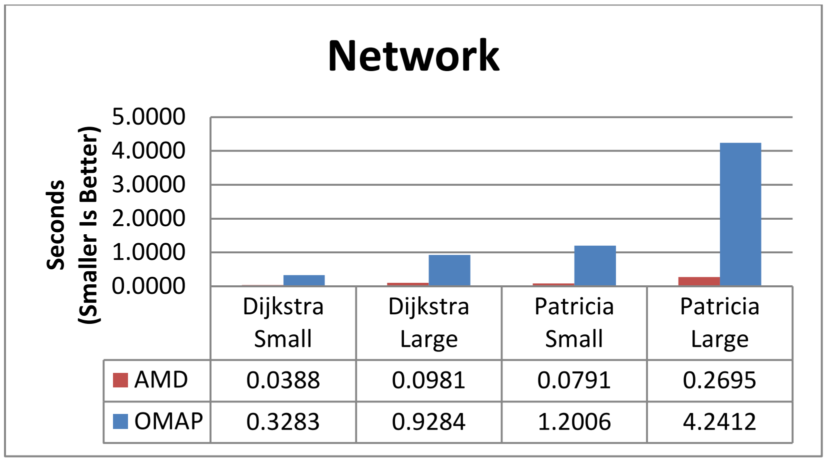


	Bitcount Small	Bitcount Large	Susan Edge Small	Susan Edge Large	Susan Corner Small	Susan Corner Large
🟥 AMD	0.0155	0.0209	0.0067	0.0546	0.0108	0.0220
🟦 OMAP	0.2493	0.0884	0.0163	0.2842	0.0140	0.2927

***Figure 17*: Automotive Runtimes 2**

Network

This is another benchmark categorization that had no faulty benchmarks within. However, there are only two of them in the category: dijkstra and patricia. Both of these output simple text files. The timing results can be seen below in *Figure 18*. All the benchmarks were very consistent.

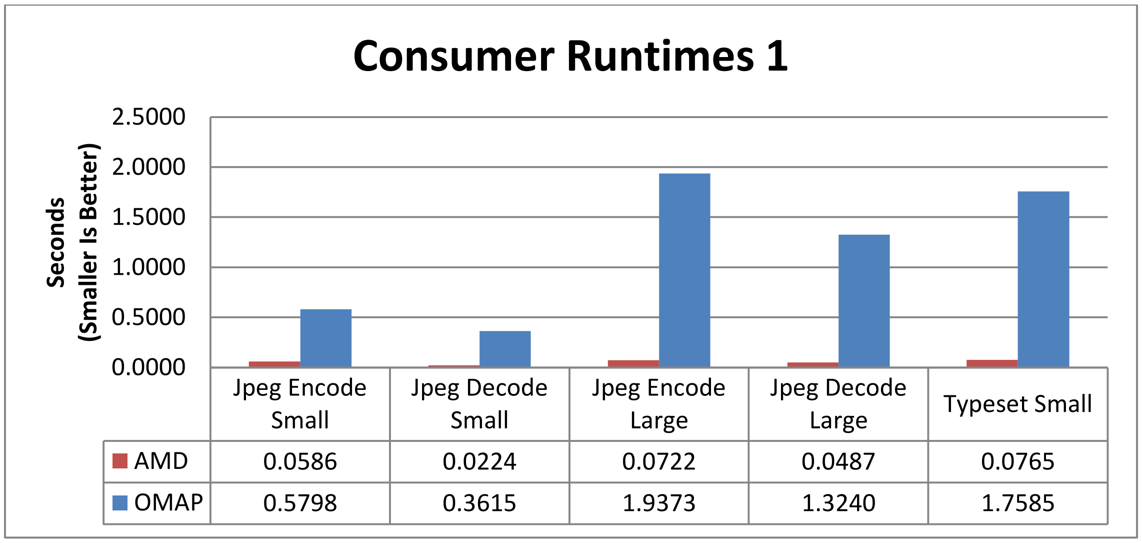


	Dijkstra Small	Dijkstra Large	Patricia Small	Patricia Large
🟥 AMD	0.0388	0.0981	0.0791	0.2695
🟦 OMAP	0.3283	0.9284	1.2006	4.2412

***Figure 18*: Network Runtimes**

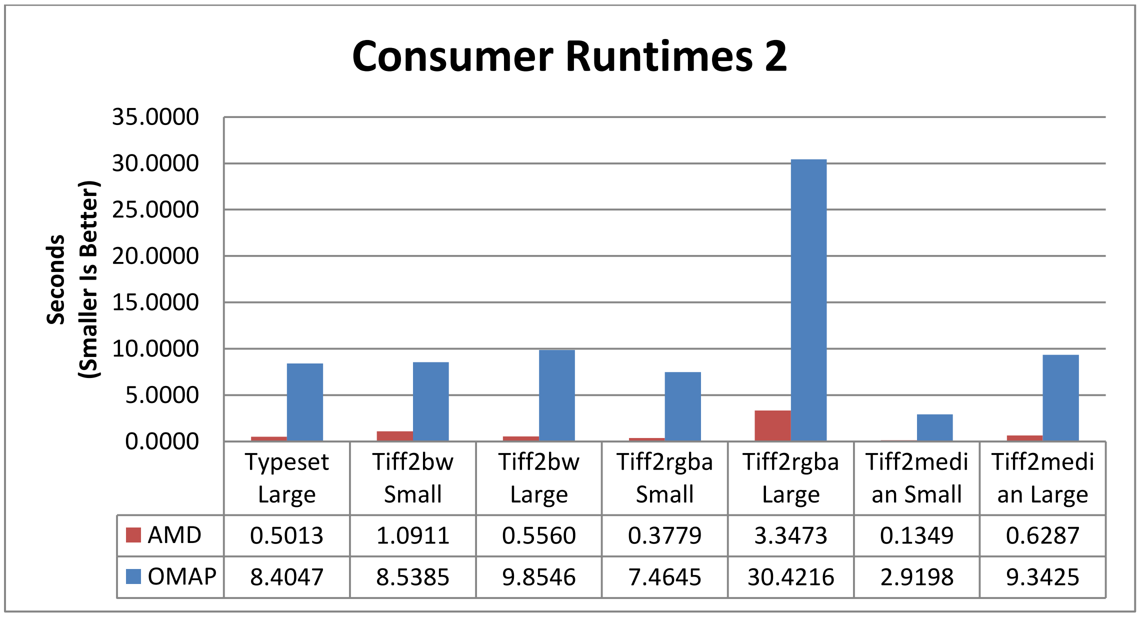
Consumer

The consumer benchmarks were faced with a large amount of difficulties during implementation. Only two of the benchmarks within, jpeg and typeset, were easy to configure and compile. Tiff, which actually has four different MiBench tests, had issues with mandatory options having been included in the makefile. Once the source of the error was found that was a simple fix. Lame compiled easily on the Beagle Board, but had issues on the other machine claiming memory addressing problems. In contrast, mad used legacy options during compilation; once fixed, it was found there were architecture specific options in some of the files that prevented getting it compiled on the Beagle Board. Once compiled, one of the tiff tests had faulty input files. The successful output consists of several images from tiff, images from JPEG, and Postscript files from typeset. The timing results from the successful tests can be seen in *Figure 19* and *Figure 20*.



	Jpeg Encode Small	Jpeg Decode Small	Jpeg Encode Large	Jpeg Decode Large	Typeset Small
🟥 AMD	0.0586	0.0224	0.0722	0.0487	0.0765
🟦 OMAP	0.5798	0.3615	1.9373	1.3240	1.7585

***Figure 19*: Consumer Runtimes**



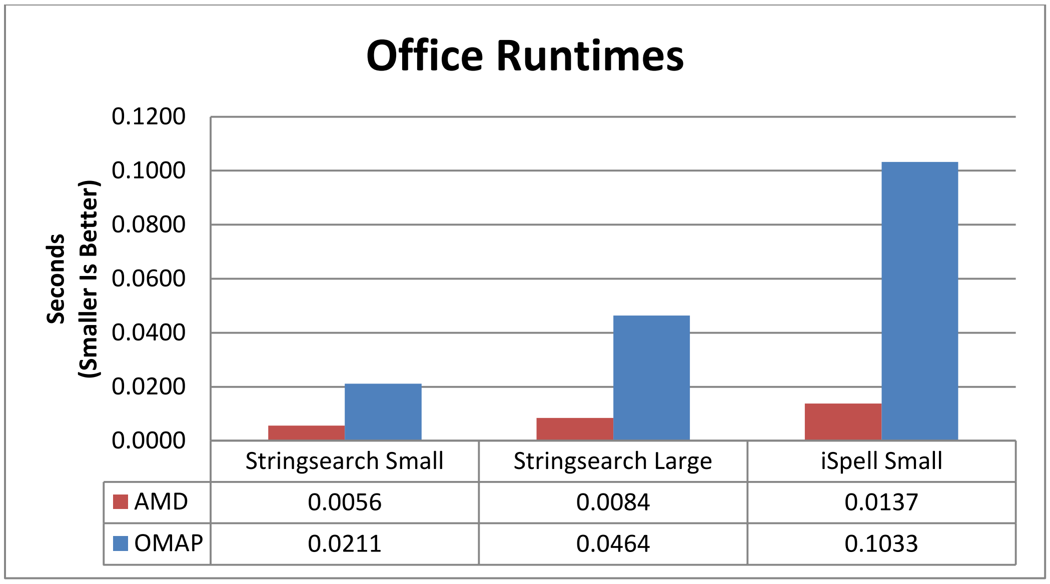
	Typeset Large	Tiff2bw Small	Tiff2bw Large	Tiff2rgba Small	Tiff2rgba Large	Tiff2median Small	Tiff2median Large
🟥 AMD	0.5013	1.0911	0.5560	0.3779	3.3473	0.1349	0.6287
🟦 OMAP	8.4047	8.5385	9.8546	7.4645	30.4216	2.9198	9.3425

***Figure 20*: Consumer Runtimes 2**

Much like in the automotive benchmarks, some of these had some anomalous results for a single test, although most did not. The Beagle Board had some large variation specifically within tiff to bw conversions. Also of note, the AMD had a vastly different runtime for the first result of some of the tiff conversion tests than the other four. This is likely due to being loaded from the hard drive the first time. Outside of these instances though, the results were fairly consistent and had a small standard deviation.

Office

This is another set of benchmarks that had far less than ideal outcome while compiling and running them; only two out of five were successfully implemented. One of them, ghostscript, was unable to compile due to poor coding in reference to library use and mistakes as amateur as having the wrong number of arguments in a function call. It was unclear how much debugging and rewriting would have been required. Rsynth was unable to even configure. Finally, sphinx, did successfully configure and compile after a large number of declaration issues as well as a pointer error that had to be debugged. Unfortunately, there were no instructions or indication of the test to be run by the program. The two functional benchmarks had text files for output. The runtime results of these are included in *Figure 21*, below. One may notice that iSpell large was not included, its results were similar to the small in ratio between machine performance, though severely impacted the scale of the chart and was not significant enough to warrant a second chart. The other results were fairly consistent with one another, though much like in some of the previous cases, the first pass of each test had an increase in magnitude of runtime for the laptop. One other case of note, the large stringsearch’s runtime on the Beagle Board was consistently approximate to one of two vastly different times.

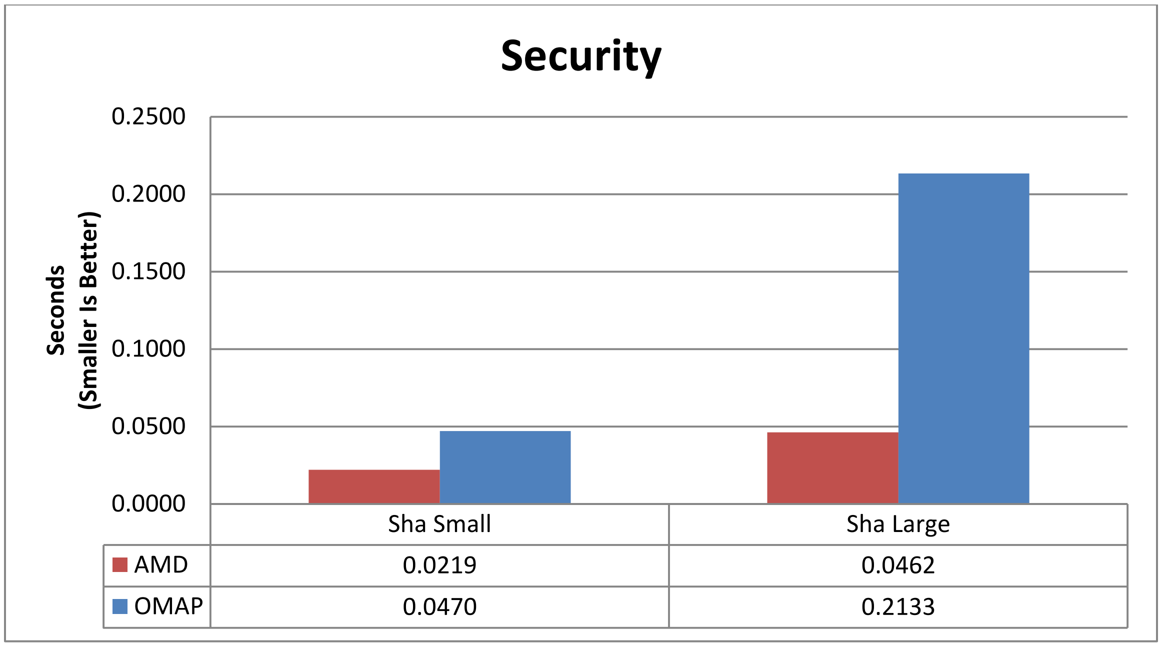


	Stringsearch Small	Stringsearch Large	iSpell Small
🟥 AMD	0.0056	0.0084	0.0137
🟦 OMAP	0.0211	0.0464	0.1033

***Figure 21*: Office Runtimes**

Security

Only one of MiBench's security oriented benchmarks were successfully implemented. The first of those that did not compile, PGP, required optimization code to be written depending on the specific operating system. As it was expecting a UNIX distribution, Ubuntu was far too removed for there to be any basis of support. When combined with the specifics of the topic of this thesis, making the results of it tangential anyway, it was deemed unnecessary to attempt to fix. Rijndael had some coding errors within. Blowfish reported some segmentation faults, and the text files came up empty. Sha successfully returned the expected text file output. A small figure (*Figure 22*) shows the runtimes of sha below.



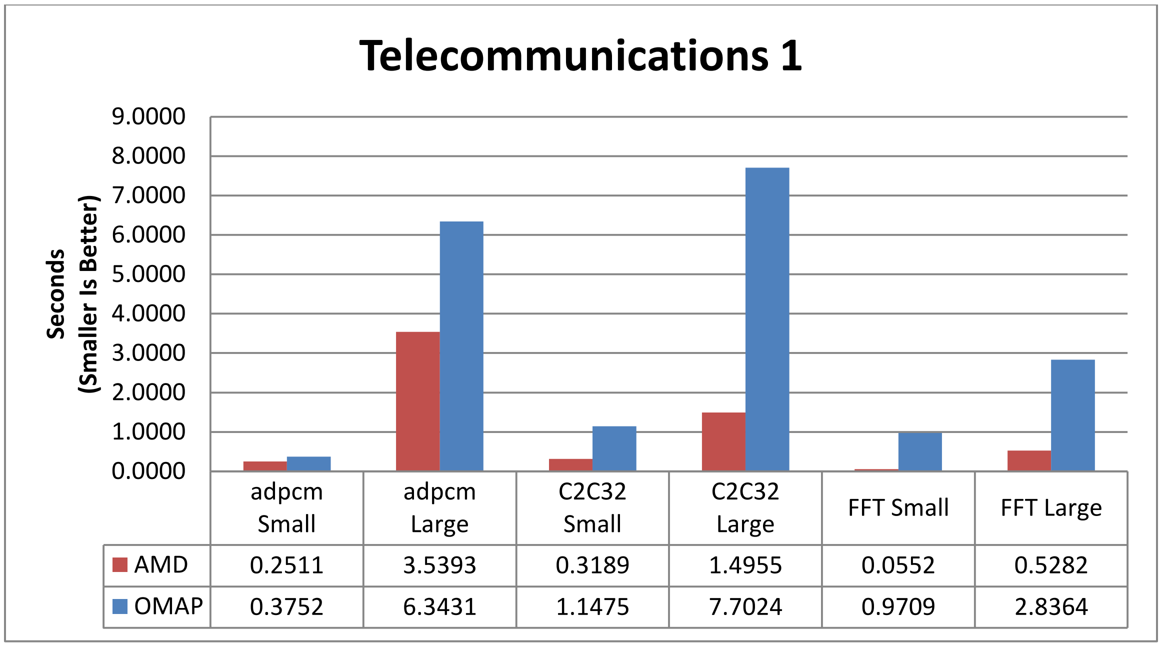
	Sha Small	Sha Large
🟥 AMD	0.0219	0.0462
🟦 OMAP	0.0470	0.2133

***Figure 22*: Security**

It should be stated that the runtimes were very consistent for the AMD processor, and the larger version of Sha as well. The smaller version had three instances that it ran slower than the worst case of the large test. This really underscores the kind of inconsistency that has been a recurring theme through the small benchmark tests.

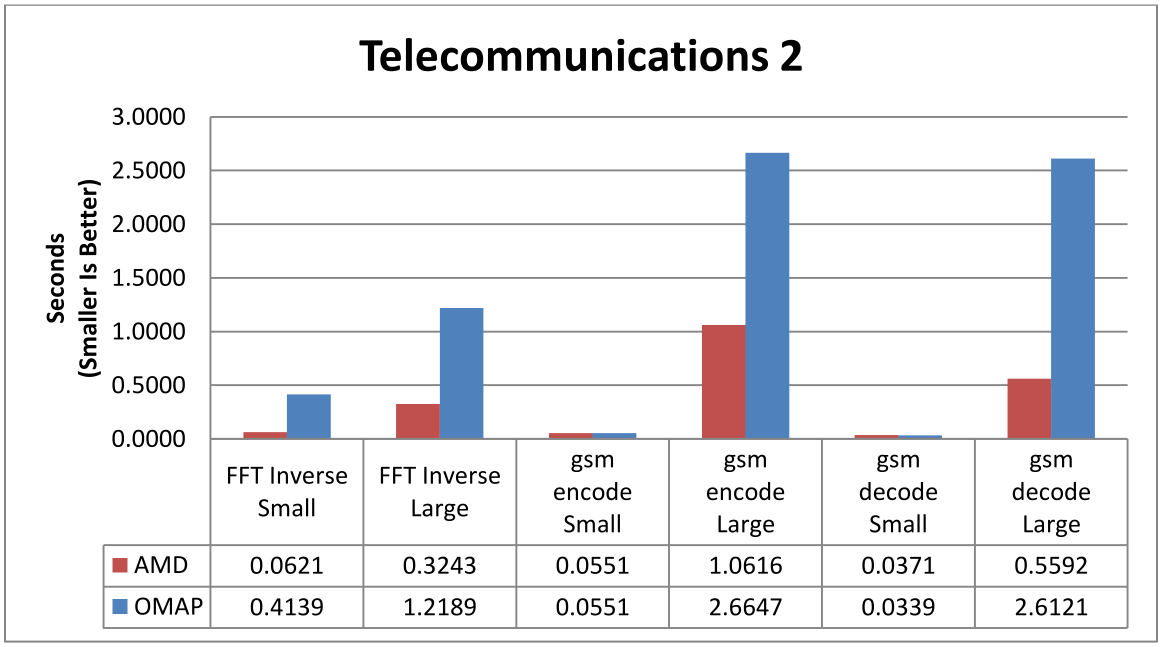
Telecommunication

All benchmarks included in telecomm were able to successfully compile; the only noteworthy complication was an ignored error out of gsm. FFT and CRC32 both had text outputs. The program gsm returned an audio output. Finally, adpcm returned a file containing pulse width modulation data. The runtimes for these programs are seen in *Figure 23* and *Figure 24*. These were actually very precise, with only a few results that had to be discounted.



	adpcm Small	adpcm Large	C2C32 Small	C2C32 Large	FFT Small	FFT Large
🟥 AMD	0.2511	3.5393	0.3189	1.4955	0.0552	0.5282
🟦 OMAP	0.3752	6.3431	1.1475	7.7024	0.9709	2.8364

***Figure 23*: Telecommunications Runtimes**



	FFT Inverse Small	FFT Inverse Large	gsm encode Small	gsm encode Large	gsm decode Small	gsm decode Large
🟥 AMD	0.0621	0.3243	0.0551	1.0616	0.0371	0.5592
🟦 OMAP	0.4139	1.2189	0.0551	2.6647	0.0339	2.6121

***Figure 24*: Telecommunication Runtimes**

V. CONCLUSIONS

The introduction begins with a series of questions pertaining to benchmarking ARM-based applications integrated systems. Consequently, the first part of the conclusion should to be to answer them. As such, the first section explores each of them independently so that the thesis statement may be objectively proven or disproven. It is at this juncture that the literature review, the benchmarks studied, and the MiBench results finally culminate.

The second section provides an overview of a few possible avenues of further research and explorations. This thesis only touches on what could be done with these devices in the realm of benchmarking. Not only could the topics within this thesis be further expanded, other related fields of study could originate from topics germane to this paper.

In the final section of this paper, the validity of the thesis statement shall be challenged. The questions answered in the beginning of this chapter become immediately and directly relevant. Upon completing the consequent analysis, the paper will begin to close with the current state of benchmarking ARM-based application integrated systems. Finally, it concludes with the reasons for this status and what (if anything) should be done to improve it.

ANALYSIS

Several questions are posed in the introduction of this thesis. The first two of these are basic; they pertain to the validity of a unique benchmark suite tailored to ARM-based applications integrated systems. The next pertinent topic to have been explored is in regard to the status of available benchmarks that applies to the target system as well as the reasons for it. Finally, an idea of the future of ARM-based technology is to be provided. In short, the answer to these questions implies that there is a need that has not been filled. The reasons why will be explored in more detail below.

Are Application Processors Distinct?

One needs to look no further than the severity that the antique AMD dominated the OMAP to clearly see the extent in which regular CPU processors can stand apart from application processors. This outcome was expected, and the inclusion of the AMD was just for scale, but that does not change the results. It could be said that the beagle board performed poorly due to bottlenecks and other influences, or that given a fair test utilizing a simulator might have made them closer. Even then, as many shared factors as possible were introduced. The clock speed advantage of the AMD made a large difference, though that underscores a difference between the processor families; thus, changing one in turn changes the characteristics. Even if one were to adjust for the fact that AMD had a clock speed three times faster than the OMAP, the AMD consistently performed at least a full increment of time faster. Common sense dictates that a modern computer versus an OMAP would have even wider margins. Speculations based off of a single test of questionable fairness are not enough to conclude where applications processors stand in the processor hierarchy however. Modern processors characterize deeper pipelines, simultaneous multithreading, and other options not yet present in most applications processors. They operate in higher frequencies, with ARM's most advanced model (See Appendix C), still operating at a max of 2 GHZ. Applications processors also still carry many expected/required traits of embedded systems that provide constraints, such as being designed to utilize lower power and require a very small circuit board footprint. One final difference, these processors are designed in such a way they do have the luxury of separate peripherals to aide in specific computations such as the video cards and sound processing available to regular processors. To illustrate this, the NEON 10 stage pipeline in the Cortex-A8 would not be found in regular CPUs. To summarize, although the differences are very thin, applications processors do have different design criteria, and are not quite as powerful as regular processors.

On the other end of the spectrum, the raw facts from the list of characteristics that distinguish the Cortex-A series from the other two ARMv7 families go a long ways to show they should be treated separately from other micro controllers. Different memory management, expansion into multi-core markets, capacity to use full operating systems, branch prediction, and deep pipelines are a few of the properties that make them stand apart. Also outside of devices using ARM's Thumb, real-time DSP and microcontrollers are not generally 32-bit cores. Simply put, for modern devices like mobile handsets, the capabilities of microcontrollers and other lower end embedded systems processors just aren't enough. Therefore, applications processors most certainly stand apart from them.

Nothing clearly outwardly states that applications processors should be treated in their own category, though logically they are aimed at a much different purpose than other embedded systems or full computers. They have different design criteria than either, and an expected performance range that really is centered between the other two. They share qualities from both categories of processor. Even if application processors do not justify specific benchmarks tailored towards them, they certainly need a suite that covers design constraints of both computers and embedded systems for wholesome evaluations.

Is Testing the Core Enough?

One of the questions in the introduction regarded whether or not testing the core was enough to distinguish between integrated systems. In this case the answer is very straight forward: no. One of the previous benchmark evaluations surveyed in *Chapter 4* clearly demonstrated that two integrated systems with similar cores (both ARM11s) performed vastly different. In that case the i.MX markedly outperformed the corresponding OMAP in almost every comparison, occasionally even before mitigating the i.MX clock speed disadvantage. This proves that two integrated systems with the same core perform differently. Another benchmark compared a vast array of different cell phones that utilized different integrated system devices, while they had different cores, most of them were all ARM based. Additionally it underscored that several competitors were using similar cores, and getting different products upon making their own modifications within the same market. Again, the components included within an integrated system make it stand apart from the core it originates from and should be considered when comparing two devices. Some of the performance differences may simply be because of the application method of that core within the integrated system, but it is likely beneficial to intentionally test and compare the components when designing a benchmark suite for them.

Are Current Benchmarks Adequate?

None of the benchmarks surveyed in this paper met the criteria of being comprehensive, holding to standard quality, and being modern enough. Some of them were closer than others, and by piecing together several of them, one might be able to comprehensively test a full processor. However, using small parts of benchmarking suites is considered misuse, and trying to use pieces of a set of benchmarks doesn't leave much room for industry wide comparisons. Essentially, there were no strong candidates for the job; in particular MiBench was a poor option for the benchmarking the target system.

MiBench was terrible for the task, but was one of the most comprehensive options. It was just too far out of date and the programs within suffered for it. Ignoring the failed compilations, it still did not have any streaming video or other benchmarks, which are largely important in today's application processors. The most detrimental feature of it was that there was not a supplied method of implementation, nor were any other benchmark results readily apparent to compare to. Because of this, a scale (the AMD) had to be included just to provide context for the results. Without it, the results would have been entirely meaningless.

Realistically, the answer as to whether or not a benchmark suite exists that is viable for the target platform cannot be exhaustively determined; thus, some amount of supposition is required. The best answer then is "not really." There are some that can provide meaningful results, particularly single purposed benchmarks as indicated by the mobile handset comparison provided in *Chapter 4*. Large portions of some of the larger suites also apply well, though none of them really distinguished themselves as the correct choice. Because of this, the final stance of this thesis regarding this particular subject is that there are enough resources available to make due for minor comparisons but nothing that comes close to being a reasonable benchmarking standard candidate for these systems.

What is the Future Trend?

ARM processors have been rapidly growing and continue to do so at an accelerated rate. With the sheer size of their portion of the market, in conjunction with how fast it got there, it is apparent that ARM will be around for some time. Major corporations that might normally develop their own processors are simply licensing ARM technology and placing them in these integrated systems. Also, processor developers are specifically targeting the application processor market that ARM currently dominates. Not only will ARM continue to expand in the foreseeable future, other comparable devices will begin to join them on the market.

Future Work

Three possible avenues of future work are readily apparent. The first of these lies in the possibility of exploring further benchmark capabilities on either the target platform or similarly related ones. This could be continuing the research done on available benchmarks. After all, while thorough, it would be naïve to claim that every possible benchmark was reviewed; just the most prominent and apparent ones were. Another worthwhile endeavor on the subject of benchmark research would be in determining precisely what an applications processor does need to be thoroughly compared. A variant of this could be studying benchmarks for integrated systems. Doing similar studies to expand outside of some of the specific traits chosen for the target system could also yield interesting results.

In an entirely different vein, the second choice for potential future studies stems from non-benchmark driven ideas. The Beagle Board alone sports many projects on its community website. Outside of those, with the right tools many programming projects are available. Studying other uses for this product could yield many project ideas. Also examining other aspects, such as design instead of testing, on integrated systems is a recommended topic for study.

Finally, one might continue the brief experiments done here; that would be much more germane to this topic than the second option for future endeavors. One very obvious way would be to compare a variety of different test boards similar to the Beagle Board, perhaps even using different revisions of the Beagle Board. For this purpose, one could use MiBench or an entirely different benchmark. A test less compromised by the real world, such as simulating these devices or using a logic analyzer on a JTAG input, could prove highly interesting as well. Finally, a project dedicated to actually developing a benchmark suite for the target platform would be worthwhile work.

SUMMARY

At the onset of this paper it was theorized that ARM-based application integrated systems were unique, yet prominent enough to warrant their own standard of benchmarking suites and that there were none readily apparent. During the course of attempting to prove or disprove this several topics were covered. First and foremost, the cores within an integrated system should not be solely consulted when examining the product. Also, application end embedded systems certainly stand apart from other processors. As ARM dominates huge portions of the market and functions on selling IP cores, their technology is almost synonymous with the field currently. Finally, a survey of benchmarks and implementation of one of them demonstrated the lack of real choices for comparison. Therefore, it is in fact valid to say benchmarks developed for ARM-based application integrated systems are lacking despite being a near necessity.

BIBLIOGRAPHY

[1] ARM Holding Ltd. ARM Architecture Overview. PDF.

[2] ARM Holding Ltd. (2011) Company Profile. [Online]. http://www.arm.com/about/company-profile/index.php

[3] ARM Holding Ltd. (2011) ARM Licensees. [Online]. http://www.arm.com/products/processors/licensees.php

[4] ARM Technology, " ARM Achieves 10 Billion Processor Milestone," Jan. 2008. [Online]. http://www.arm.com/about/newsroom/19720.php

[5] ARM Holding Ltd. (2011) Milestones. [Online]. http://www.arm.com/about/company-profile/milestones.php

[6] P. Rickert and W. Krenik. (2006) Cell Phone Integration: SiP, SoC, and PoP. pdf.

[7] J.-Q. Lu. (2009, Feb.) 3-D Hyperintegration and Packaging Technologies for Micro-Nano Systems. pdf.

[8] R. Weeraskera, D. Pamunuwa, L.-R. Zheng, and H. Tenhunen, "Two-Dimensional and Three-Dimensional Integration of Heterogeneous Electronic Systems Under Cost, Performance, and Technological Constraints," *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, vol. 28, no. 8, Aug. 2009.

[9] T. Spits and P. Werp. (2000) OMAP Technology Overview. pdf.

[10] M. R. Guthaus, et al. (2001, Dec.) MiBench: A free, commercially representative embedded benchmark suite. [Online]. http://www.eecs.umich.edu/mibench/Publications/MiBench.pdf

[11] C. A. a. A. v. Someren, "The History of the ARM CPU," in *'The ARM RISC Chip: A Programmers' Guide*. Addison-Wesley, 1993. [Online]. http://www.ot1.com/arm/armchap1.html

[12] T. Krazit, "ARMed for the living room ," *CNET news*, Apr. 2006. [Online]. http://news.cnet.com/ARMed-for-the-living-room/2100-1006_3-6056729.html

[13] ARM Holding Ltd. (2011) DSP & SIMD. [Online]. http://www.arm.com/products/processors/technologies/dsp-simd.php

[14] ARM Holding Ltd. (2011) Neon. [Online]. http://www.arm.com/products/processors/technologies/neon.php

[15] ARM Holding Ltd. (2011) TrustZone. [Online]. http://www.arm.com/products/processors/technologies/trustzone.php

[16] ARM Holding Ltd. (2011) Instruction Set Architectures. [Online]. http://www.arm.com/products/processors/technologies/instruction-set-architectures.php

[17] L. Devices, "ARM aims son of Thumb at uCs, ASSPs, SoCs," ***Linuxfordevices.com*****, Oct. 2004. [Online]. http://www.linuxfordevices.com/c/a/News/ARM-aims-son-of-Thumb-at-uCs-ASSPs-SoCs/**

[18] ARM Holding Ltd, "New ARM Thumb-2 Core Technology Provides Industry-Leading Levels Of Code Density And Performance," Jun. 2003. [Online]. http://www.arm.com/about/newsroom/319.php

[19] ARM Holding Ltd. (2011) Jazelle. [Online]. http://www.arm.com/products/processors/technologies/jazelle.php

[20] ARM Holding Ltd. (2011) ARM Glossary. [Online]. http://www.arm-development.com/arm_glossary

[21] ARM Holding Ltd. (2011) Processors. [Online]. http://www.arm.com/products/processors/

[22] ARM Holding Ltd. (2011) Classic - ARM 7 Profile. [Online]. http://www.arm.com/products/processors/classic/arm7/index.php

[23] ARM Holding, Ltd. (2011) Classic - ARM 9 Profile. [Online]. http://www.arm.com/products/processors/classic/arm9/index.php

[24] ARM Holding Ltd. (2011) Classic - ARM 11 Profile. [Online]. http://www.arm.com/products/processors/classic/arm11/index.php

[25] ARM Holding Ltd. (2011) Cortex-R Series. [Online]. http://www.arm.com/products/processors/cortex-r/index.php

[26] ARM Holding Ltd. (2011) Cortex-M Series. [Online]. http://www.arm.com/products/processors/cortex-m/index.php

[27] ARM Holding Ltd. (2011) Cortex-A Series. [Online]. http://www.arm.com/products/processors/cortex-a/index.php

[28] W. Hohl, ***ARM Assembly Language*****. Boca Raton, Florida, USA: CRC Press, 2009.**

[29] J. Lee and H.-J. Lee, "Wire Optimization for Multimedia SoC and SiP Designs," ***IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I*****, vol. 55, no. 8, Sep. 2008.**

[30] Texas Instruments. (2011) OMAP 3530. [Online]. http://focus.ti.com/docs/prod/folders/print/omap3530.html

[31] A. Rougier and E. Bagerman, "SiC and SoC will remain Co-existing System Solutions," ***Solid State Technology*****, p. 36, Oct. 2007.**

[32] A. Maurelli, D. Belot, and G. Campardo, "SoC and SiP, the Yin and Yang of the Tao for the New Electronic Era," ***Proceedings of the IEEE*****, vol. 97, no. 1, Jan. 2009.**

[33] A. Hardy, "What Powers Android? Hummingbird vs. Snapdragon vs. OMAP vs. Tegra 2: ARM Chips Explained," Jan. 2011. [Online]. http://androidheadlines.com/2011/01/what-powers-android-hummingbird-vs-snapdragon-vs-omap-vs-tegra-2-arm-chips-explained.html

[34] Freescale Semiconductor. (2011) IMX Home. [Online]. http://www.freescale.com/webapp/sps/site/homepage.jsp?code=IMX_HOME

[35] R. Allan, "IC Packages Feel the Squeeze," *Electronic Design*, Oct. 2007.

[36] R. Crisp. (2008, Oct.) EE Times. [Online]. http://www.eetimes.com/electronics-news/4079408/killer-app-for-cell-handsets

[37] F. Carson, "Innovations Push Package-on-Package Into New Markets," *Semiconductor International*, Apr. 2010.

[38] Beagleboard.org. (2009, Dec.) BeagleBoard System Reference Manual REV C4. [Online]. http://beagleboard.org/static/BBSRM_latest.pdf

[39] Free Software Foundation, INC. (2011) GNU Operating System. [Online]. http://www.gnu.org.

[40] G. Coley, "Take advantage of open-source hardware," *EDN*, vol. 54, no. 16, pp. 20-23, Aug. 2009. [Online]. http://www.proquest.com.argo.library.okstate.edu

[41] R. Paul, "TI launches hackable Beagle Board for hobbyist projects," *Ars Technica*, Aug. 2008. [Online]. http://arstechnica.com/open-source/news/2008/08/ti-launches-hackable-beagle-board-for-hobbyist-projects.ars

[42] Digi-Key Corporation, "USB-powered Beagle Board from Digi-Key Unleashes Community Development with Laptop-like Performance and Expansion for $149.," *Journal of Engineering*, p. 48, Aug. 2008. [Online]. http://www.proquest.com.argo.library.okstate.edu

[43] beagleboard.org. (2011) Beagleboard. [Online]. http://beagleboard.org/

[44] Liquidware. Liquidware - BeagleTouch. [Online]. http://www.liquidware.com/shop/show/BB-BT/BeagleTouch

[45] Liquidware. (2010) Liquidware sales: Beaglejuice. [Online]. http://www.liquidware.com/shop/show/BB-BJC/BeagleJuice

[46] Tincan Tools. (2011) Flyswatter Sales. [Online]. http://www.tincantools.com/product.php?productid=16134&cat=0&page=1&featured

[47] ARM Holding Ltd. (2011) ARM Cortex-A8. [Online]. http://arm.com/products/processors/cortex-a/cortex-a8.php

[48] Embedded Insights. (2010) Cortex-A8. [Online]. http://www.embeddedinsights.com/epd/arm/arm-cortex-a8.php

[49] A. L. Shimpi. (2011) NVIDIA's Tegra 2 Take Two: More Architectural Details and Design Wins. [Online]. http://www.anandtech.com/show/4098/nvidias-tegra-2-take-two-more-architectural-details-and-design-wins/2

[50] Embedded Developer. (2006) ARM Cortex A8. [Online]. http://www.embeddeddeveloper.com/cores_variant/14/ARM-Cortex-A8.htm

[51] Texas Instruments. (2011) Cortex-A8_Architecture. [Online]. http://processors.wiki.ti.com/index.php/Cortex-A8_Architecture

[52] ARM Holding Ltd. (2011) ARM Cortex-A8 Reference Manual. PDF. [Online]. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344k/DDI0344K_cortex_a8_r3p2_trm.pdf

[53] D. Citron. (2003) MisSPECulation: Partial and Misleading Use of SPEC CPU2000 in Computer Architecture Conferences. PDF.

[54] A. R. Weiss. (2002, Nov.) Dhrystone Benchmark White Paper.

[55] Standard Performance Evaluation Corporation. (2011, Apr.) SPEC Corporation Webiste. [Online]. http://www.spec.org/

[56] The Embedded Microprocessor Benchmark Consortium. (2011) EEMBC Corporate Website. [Online]. http://www.eembc.org/home.php

[57] BDTI. (2011) BDTI Corporate website. [Online]. http://bdti.com/

[58] L. K. John. (2006) Performance Evaluation: Techniques, Tools and Benchmarks. pdf. [Online]. http://www.ece.utexas.edu/~ljohn

[59] Synchromesh Computing Services . (2006) Synchromesh Computing Benchmarking: Freescale i.MX31, TI OMAP 2420, Intel Bulverde for Control Code, “Out of the Box”. Powerpoint.

[60] K. Williston, "Benchmarking basics, part 2: BDTI and EEMBC reviewed," Jul. 2008. [Online]. http://www.eetimes.com/design/signal-processing-dsp/4017669/Benchmarking-basics-part-2-BDTI-and-EEMBC-reviewed

[61] BDTI. (2011) BDTI Cortex-A8 Overview. [Online]. http://www.bdti.com/Resources/BenchmarkResults/Processors/Cortex-A8

[62] EEMBC. (2011) EEMBC main page. [Online]. http://www.eembc.org/home.php

[63] K. Williston. (2008, Jun.) EE Times. [Online]. http://www.eetimes.com/design/signal-processing-dsp/4017666/Benchmarking-basics-part-1-Choosing-and-using-benchmarks

[64] S. Gal-On and M. Levy, "CoreMark: A realistic way to benchmark CPU performance," Jan. 2011. [Online]. http://www.eetimes.com/design/embedded/4212735/CoreMark--A-realistic-way-to-benchmark-CPU-performance

[65] J. Fritts. MediaBench Consortium. [Online]. http://euler.slu.edu/~fritts/mediabench/

[66] Ubuntu. (2011) Ubuntu Homepage. [Online]. http://www.ubuntu.com/

[67] CJLippstreu. (2011, Feb.) Smart Ketai. [Online]. http://www.smartkeitai.com/lg-optimus-3d-beats-samsung-galaxy-s-ii-in-benchmarks/

[68] K. Roberts-Hoffman and P. Hegde. (2009) ARM Cortex-A8 vs. Intel Atom: Architectural and Benchmark Comparisons. pdf.

[69] Synchromesh Computing, LLC. Synchromesh Computing website. [Online]. http://www.synchromeshcomputing.com/index.php

[70] Synchromesh Computing, LLC. (2004) Evaluation and Benchmark Testing: The Freescale Semiconductor i.MX21 Processor. [Online]. http://brianrwright.com/BrianRWright/Assets/Files/White_Paper_imx21_rev2.pdf

[71] EEMBC. (2011) Cormark Org website. [Online]. http://www.coremark.org/home.php

[72] J. Turley, "ARM’s Cortex-A15 “Eagle” Has Landed," Nov. 2010. [Online]. http://www.eejournal.com/archives/articles/20101109-cortex/

[73] Sunsider developers. Sunspider Website. [Online]. http://www.webkit.org/perf/sunspider/sunspider.html

[74] Rightware . Browsermark . [Online]. http://www.rightware.com/en/Benchmarking+Software/BrowserMark/

[75] GLBenchmark. (2011) GLBenchmark Home Page. [Online]. http://www.glbenchmark.com/

[76] AMD. (2006, Sep.) AMD Turion™ 64 X2 Mobile Data Sheet. [Online]. http://support.amd.com/us/Processor_TechDocs/41407.pdf

[77] ARM Holding Ltd. (2011) Cortex-A9 White Paper. [Online]. http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf

[78] ARM Holding Ltd. (2011) ARM Cortex-A5 Reference Manual. pdf.

[79] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor System-on-Chip (MPSoC) Technology," *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, vol. 27, no. 10, Oct. 2008.

[80] HINT . (1999) HINT Main Page. [Online]. http://hint.byu.edu/

[81] J. D. McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. [Online]. http://www.cs.virginia.edu/stream/

[82] Inquisitor. (2008) Inquisitor BYTEmark Profile . [Online]. http://www.inquisitor.ru/doc/tests/bytemark.html

GLOSSARY

ARM Holding Ltd – British company that owns and develops the ARM instruction set, and sells ARM cores as intellectual property

Advanced RISC Machine (ARM) – an embedded instruction set architecture that is based on and expands upon reduced instruction set computer design strategies

Applications Processor – embedded processor that is used in high end devices, and is capable of running a full operating system

Beagle Board – hobbyist device with large community support that was designed by Texas Instrument’s engineers to allow users to experiment with the OMAP3530

i.MX – An Intel SoC that utilizes ARM ISA, and one of the competitors to OMAP

Integrated System – Any device that contains multiple system components within a single package so that might occupy a single footprint such as SoCs, SiPs, and PoPs

IP Cores – The intellectual content of a processor core design to be licensed on its own so that the licensee may modify, expand, integrate, and manufacture it to their needs

Jazelle – A specific ISA expansion for ARM for Java Bytecode

MiBench – A generic open-source embedded systems benchmark suite

NEON – A media oriented ISA expansion for ARM, occasionally implemented with its own pipeline

OMAP –A Texas Instruments SoC family of devices that contains an ARM application core and is commonly used in mobile handheld devices

Package – The ceramic or plastic casing around a chip(s) with different I/O leads for connecting the contained device to other components

Package-on-Package – A method of stacking different packages and interfacing them directly with one another

System-in-Package – A package that contains multiple chips, either on the same plane or stacked on top of one another

System-on-a-Chip – A chip/die that has multiple system components printed on it

Thumb – A special ISA expansion to be included with ARM that uses 16-bit instructions and expands them to the 32-bit registers seamlessly

ACRONYMS

AOT – Ahead-of-Time

ARM – Advanced RISC Machine

ASICs – Application Specific Integrated Circuits

BOM – Bill of Materials

CISC – Complex Instruction Set Computers

CPSR – Current Program Status Register

DBX – Direct Bytecode eXecution

i.MX – innovative Multimedia eXtension

IP – Intellectual Property

ISA – Instruction Set Architecture

OMAP – Open Multimedia Application Platform

JIT – Just-in-Time

MMU – Memory Management Unit

MPU – Memory Protection Unit

PCB – Printable Circuit Board

PoP – Package on Package

PSoC – Programmable System on Chip

RCT – Runtime Compilation Target

RISC – Reduced Instruction Set Computer

RTOS – Real Time Operating System

SDHC – Secure Digital High Capacity

SIMD - Single Instruction Multiple Data

SiP – System in Package

SoC – System-on-a-Chip

SPSR – Saved Program Status Registers

UAL – Universal Assembly Language

APPENDICES

Appendix A: Partition Geometry

To establish Ubuntu in a way that the Beagle Board is ready to run it from an SD card, it is necessary to correctly set up the partition geometry. Before creating the correct partitions it is required to format the device and clear the partition table. Once confirmed that the device has no established partitions then the process can begin. The number of heads, sectors, and cylinders must be set. First, the number of heads is set to *255* and the number sectors are set to *63*. This is constant no matter what SD card is chosen. The formula below is used to calculate the number of cylinders.

$$C = \frac{\frac{\frac{B}{255}}{63}}{512}$$

The equation uses C to represent the number of cylinders and B to represent the number of bytes on the SD card. The values come from the number of heads, sectors, and by assigning 512 bytes per sector. The result of C should be rounded down to the nearest integer. Once the geometry is established, the only thing left to do is to create the individual partitions. The first partition is the FAT32, which is placed on the first *50* cylinders of the card and marked as bootable. The remaining cylinders are used for the ext2 partition, which contains the root file system.

Appendix B: MiBench Details

This appendix provides two important collections of information relevant to the MiBench [10] benchmark suite. Primarily it provides a rough description of each test or algorithm included within the suite. These are broken into six tables (*Table 7*-

Table 12), one for each category of benchmark. *Figure 25* is an image of a table from the MiBench summary [10] that provides the instruction counts for each of the tests. This information is secondary to the descriptions.

***Table 7*: Automotive & Industrial Benchmark Descriptions**

Program	Description
basicmath	Basic math that wouldn't require dedicated hardware: Integers, angles, etc
bitcount	Tests bit manipulation abilities using different methods
qsort	Sorts array of strings
susan (edges)	Recognizes edges in an images
susan (corners)	Recognizes corners in an images
susan (smoothing)	Smooths edges in an image

***Table 8*: Consumer Benchmark Descriptions**

Program	Description
jpeg	Compresses and decompresses images
lame	Encodes MP3 format sound waves
mad	MPEG audio decoder
tiff2bw	Converts colored tiff image to black and white
tiff2rgba	Converts colored tiff image to RGB format
tiffdither	Used for reduction of size and resolution of an image
tiffmedian	Simplifies/reduces color palette of image
typeset	Emulates typesetting an HTML file

Table 9: **Office Benchmark Descriptions**

Program	**Description**
ghostscript	Used to interpret postcript
ispell	Spell Checker
rsynth	Text to speech
sphinx	Speech decoder
stringsearch	Case senstive search for strings

Table 10: **Network Benchmark Descriptions**

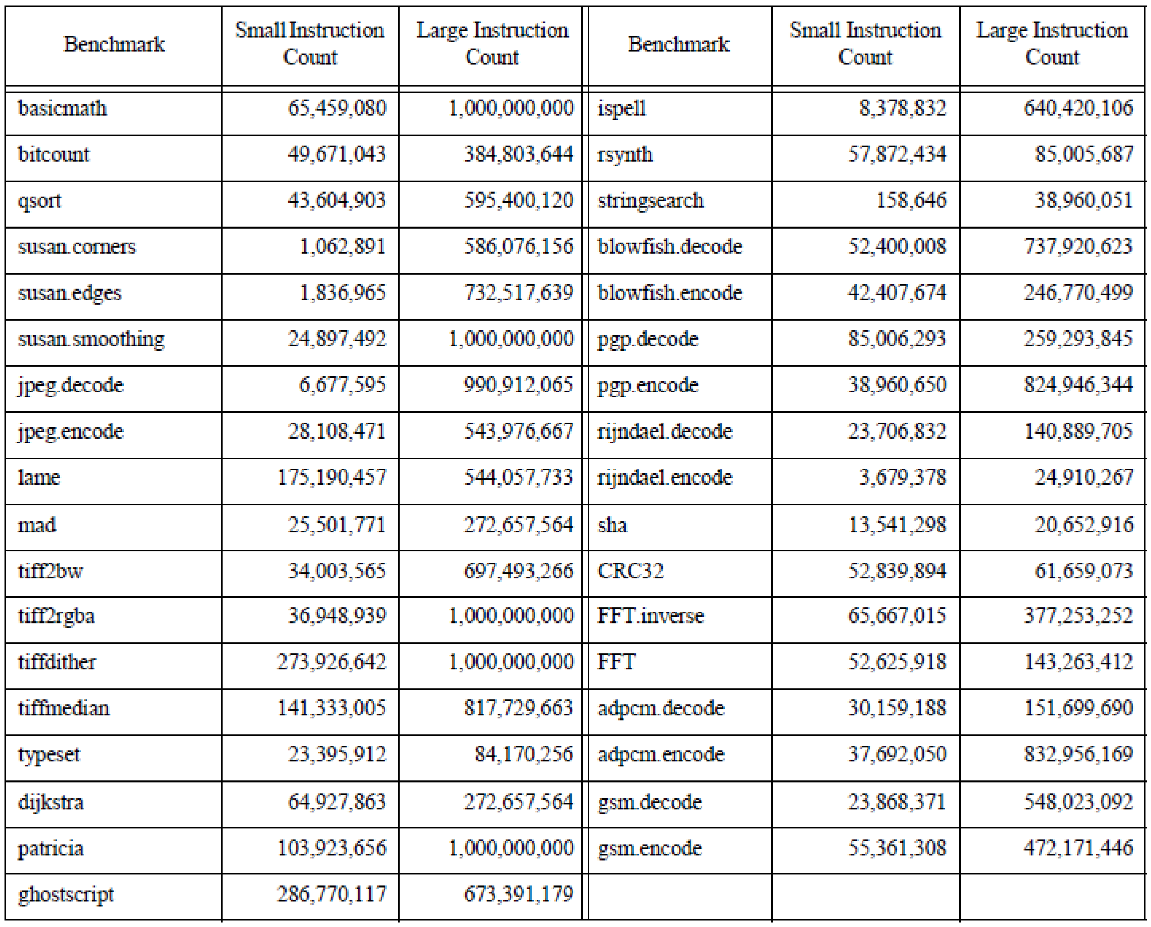
Program	**Description**
dijkstra	Calcuates shortest path in adjacency matrix
patrica	Used to represent routing tables using trees
(CSC32)	See Telecommunications Table
(sha)	See Security Table
(blowfish)	See Security Table

Table 11: **Security Benchmark Descriptions**

Program	**Description**
blowfish enc/dec	Symmetric block cipher with variable length key
pgp sign/verify	Public encryption key for secure communication
rijndael enc/dec	Block cipher; an encryption standard
sha	Secure hash algorithm for exchanging crytographic keys

Table 12: **Telecommunications Benchmark Descriptions**

Program	**Description**
CRC32	Checksum program, 32-bit cyclic redundancy check
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
ADPCM enc/dec	Adaptive Differential Pulse Code Modulation
GSM enc/dec	Voice encoding and decoding



Benchmark	Small Instruction Count	Large Instruction Count	Benchmark	Small Instruction Count	Large Instruction Count
basicmath	65,459,080	1,000,000,000	ispell	8,378,832	640,420,106
bitcount	49,671,043	384,803,644	rsynth	57,872,434	85,005,687
qsort	43,604,903	595,400,120	stringsearch	158,646	38,960,051
susan.corners	1,062,891	586,076,156	blowfish.decode	52,400,008	737,920,623
susan.edges	1,836,965	732,517,639	blowfish.encode	42,407,674	246,770,499
susan.smoothing	24,897,492	1,000,000,000	pgp.decode	85,006,293	259,293,845
jpeg.decode	6,677,595	990,912,065	pgp.encode	38,960,650	824,946,344
jpeg.encode	28,108,471	543,976,667	rijndael.decode	23,706,832	140,889,705
lame	175,190,457	544,057,733	rijndael.encode	3,679,378	24,910,267
mad	25,501,771	272,657,564	sha	13,541,298	20,652,916
tiff2bw	34,003,565	697,493,266	CRC32	52,839,894	61,659,073
tiff2rgba	36,948,939	1,000,000,000	FFT.inverse	65,667,015	377,253,252
tiffdither	273,926,642	1,000,000,000	FFT	52,625,918	143,263,412
tiffmedian	141,333,005	817,729,663	adpcm.decode	30,159,188	151,699,690
typeset	23,395,912	84,170,256	adpcm.encode	37,692,050	832,956,169
dijkstra	64,927,863	272,657,564	gsm.decode	23,868,371	548,023,092
patricia	103,923,656	1,000,000,000	gsm.encode	55,361,308	472,171,446
ghostscript	286,770,117	673,391,179			

***Figure 25*: MiBench Instruction Counts [10]**

APPENDIX C: CORTEX-A COMPARISONS

The commonalities of the application processors are discussed in the *Background* section about ARM families; this appendix focuses on what separates them into unique purposes. Each of the primary Cortex-A cores have distinct purposes and some very significant design variations including pipeline size, multicore possibility, and ISA expansion options. There are four different Cortex-A processor cores: Cortex-A5, Cortex A8, CortexA9, and Cortex A15 [27]. The Cortex-A5 is the lowest end application processor, focusing on using low-power and being low-cost. It comes in both single and multi-core varieties. On the other end, the A15 is the high-end, high-frequency option. The Cortex-A8 is specifically focused on high-end media applications, thus it automatically includes NEON. However it only features a single core. Finally the Cortex-A9 is focused on optimizing the efficiency of power-usage to performance. It also has multi-core capability. A9 also features a hard-macro implementation that allows extremely high operating frequency. A compilation [27] [52] [77] [72] [78] of features and specifications is included in *Table 13*.

***Table 13*: Cortex-A Comparisons**

	Cortex-A5	Cortex-A8	Cortex-A9	Cortex-A15
Multicore	1 - 4, or Single	No, or Single	1 - 4, or Single	1-4X SMP
Operating Frequency Range	300-800 MHZ	600 - 1000 MHZ	600 - 2000 MHZ	1 - 2 GHZ
NEON	Optional	Included	Optional	Included
FPU	VFPv3 Optional	VFPv3 included	Optional	VFPv4 included
Pipeline Stages	8	13	8	15-24
Out of Order?	No	No	Yes	Yes
L1 Cache Size (I/D)	4-64KB/4-64KB	32-64KB	32KB/32KB	32KB/32KB
L1 Cache Associativity (I/D)	2-way/4-way	4-way	4-way	?

VITA

Seth Williams

Candidate for the Degree of

Master of Science

Thesis: BENCHMARKING ARM-BASED APPLICATION INTEGRATED SYSTEMS

Major Field: Electrical Engineering

Biographical:

Personal Data: Born son to Leo Max. Williams and Teresa Jean Williams in Lawton, Oklahoma on March 6, 1986.

Education: Graduated from Frederick High School, Frederick, Oklahoma in May 2004; received Bachelor of Science in Electrical Engineering (with Computer option) from Oklahoma State University, Stillwater, Oklahoma, in May 2009; Completed requirements for Master of Science degree with major in Electrical Engineering from Oklahoma State University in December 2011.

Experience: Raised in Frederick, Ok; employed by Oklahoma State University as Undergraduate Research Assistant May 2006 – February 20\

Name: Seth Williams Date of Degree: December 2011

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: BENCHMARKING ARM-BASED APPLICATIONS INTEGRATED SYSTEMS

Pages in Study: 87 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: This study had the primary goal of raising awareness to the availability, or lack thereof, of a benchmark suite that comprehensively, effectively, and efficiently tests ARM based system-in-package applications processors. This was done by researching standard benchmarks to determine applicability and running the optimal option on the BeagleBoard; an open-source development tool using a modern ARM processor.

Findings and Conclusions: While there were many options available, it was determined there is no testing suite has been specifically developed for this due to several reasons. First, the rate of expansion in ARM's share of the market has been rapid, indicating that a specific suite would not have been necessary even five years ago. Second, because the ARM cores are entirely licensed as IP to different competitors, comparisons between the specific processors exist already to licensees. Finally, many of the existing suites contain large portions of desired tests. It has also been found that industry suites are not specifically focused in this area. Suites that are applicable and more readily available to open source oriented communities are found to be outdate. It is the conclusion of this paper that enough resources exist to marginally compare two ARM applications processors that have been included in different packages, though the reality is far from ideal.

ADVISER'S APPROVAL: James Stine