HIGH PERFORMANCE REAL-TIME EMBEDDED SYSTEMS: DESIGN AND IMPLEMENTATION OF ROAD SURFACE ANALYZER SYSTEM

by

JAREER HAMZAH ALI ABDEL QADER

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2010

This dissertation is dedicated to my mother and father

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Prof. Roger Walker whose close supervision, deep knowledge, patience, encouragement, and support were the main drive behind the success of my research.

I also want to thank the staff and student members of the Transportation Instrumentation lab for their support. I want to thank Texas Department of Transportation for funding most of this research, along with partial funds from Intel for the EP80579 SOC system development.

I would like to thank the committee members, Dr. David Kung, Dr. Farhad Kamangar, Dr. Jean Gao, and Dr. Hao Che for their valuable guidance and comments.

I thank Dr. Emmanuel Fernando and colleagues at the Texas Transportation Institute, Texas A&M University, College Station, Texas for his providing data for experimental studies.

I am very grateful to my mother and father for their prayers, worries, and their endless support during my graduate studies; and my Brother Gaith, his wife Glenda and their kids Jay, Zane, and Aleaha for their support during my study, along with my sister Jenan, and my sister Jomana, her husband Sami and their son Ramsey. I also would like to extend my gratitude to my uncle Nazeeh Shalbak.

I am thankful to my friends specially Dr. Youssif Al-Nashif and Dr. Iyad Alfaluji for their encouragement.

Finally, I'm graceful to Sherwin and Dorothy Scott, Sylvia Bowen and their extended families for being my family during my study in the UTA.

April 16, 2010

ABSTRACT

HIGH PERFORMANCE REAL-TIME EMBEDDED SYSTEMS: DESIGN AND IMPLEMENTATION OF ROAD SURFACE ANALYZER SYSTEM

Jareer Hamzah Ali Abdel Qader, PhD

The University of Texas at Arlington, 2010

Supervising Professor: Roger Walker

Multi-core processors are becoming main components for many embedded systems. Designing such embedded systems is a complex task, since developers must deal with a number of issues such as multi-threading and optimal use of parallel processing. Multi-core processors can provide benefits to embedded applications in terms of performance improvement and power utilization. Problems related to real-time embedded systems in terms of designing, modeling, simulating, and implementing both hardware and software parts of embedded systems will be researched and discussed in this work.

This research investigates and establishes the necessary steps to design, analyze, and implement a real-time embedded multi-core application. The application used is a real-time road surface analyzer system. The system is used for measuring, collecting, processing, and displaying pavement surface characteristics. This application is based on the road profiler, one of the main instruments used by transportation engineers to test road surfaces and determine their condition. The research analyzes the requirements of the system, defining and designing the appropriate tools needed for implementing the real-time multi-core embedded system.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

This research effort addresses many of the complexities involved in developing a real-time multi-core embedded system. Recently, increased performance in computing components is being achieved through the use of multi-core processors. In order to make use of the full power of these devices, multi-threading and parallel computing concepts are required. Parallel processing, once only prevalent in large servers and other intensive computing machines, is now beginning to appear not only in most servers, desktop and notebook computers, but also in embedded systems. Although opening up a host of new capabilities and application for the small embedded processor, proper use of these devices present many challenges to the embedded designer. This research addresses many of these challenges, using a road profiling system as a case study.

The road profiler is an instrument used by transportation engineers for accurate measurement of road or pavement profile. From this profile various statistical variables can be computed, providing the condition or serviceability of the pavement. A road profiler is analyzed, redesigned, and implemented from a computer engineering perspective and as an embedded application. Software engineering tools were used to model the system and define the main tasks and the dependence among the tasks.

Several embedded multi-core processors were evaluated in order to select which one provides the optimal performance for this application. Then embedded operating systems were investigated to select the appropriate one to be used in this application which is capable of meeting the real-time constraints and is compatible with multi-core processors for parallel computation.

This research also relies on a number of signal processing concepts and tools for analyzing the tasks and algorithms implemented, determining the settings of the data acquisition unit, and selecting the appropriate filters in terms of type, size, and coefficients for the tasks.

1.2 Real-time Embedded Systems

An embedded system is a system that interacts with the real world. It typically reads inputs from sensors, performs some computation, and then outputs data.

Embedded systems differ from normal computer systems (desktops, laptops, servers, etc) since they are designed to run and support few applications (mostly one integrated application) with real time deadline restrictions. Factors such as power usage and system cost are of paramount concern.

Recently, parallel processing became common practice and a necessity with the advent and the spread of processors that support multi-threading by the aid of either multi-core processors, simultaneous multithreading known as the hyper-threading by Intel, or both technologies. The multi-core processors are now widely used and are becoming the essential part of all desktop systems and are beginning to be considered in the design of new embedded systems or in upgrading existing ones.

The use of parallel processing with the aid of the new processors in embedded systems add complexity to such systems in stating the way a problem is solved. Also, the application to be programmed using multi-threading taking into consideration the embedded application restrictions, and which part of that system will execute which thread. The desired embedded system application should not only function correctly, but should also satisfy time constraints. It is desirable for an embedded system to provide services only at the right time.

The traditional design flow for mixed hardware-software embedded systems is to select architecture, decide where in the architecture different pieces of the application will be implemented, design the needed hardware and software pieces, and finally integrate the pieces of the system together.

1.3 Contributions of the Dissertation

This research illustrates, using a road profiler analyzer as a case study, the procedures in designing and implementing a real-time embedded system. It includes, design, system analysis, and development of new concepts.

The system used as a case study consists of a series of independent low-power general purpose instrument modules for measuring various pavement performance characteristics. Each measurement module acquires, processes, synchronizes, and communicates data between itself and other modules from one or more sensors in real-time. The sensors used in the modules include such devices as gyros, accelerometers, lasers, infrared detectors, etc.

Currently, the main TIL lab task is to upgrade the existing system and integrate new algorithms into the system along with reporting results in real time. This requires using one or more high performance multi-core embedded control processors. Such a complex system requires careful redesign by stating the tasks to be carried by the system in real-time and the requirements for every task in terms of amount of data and computational power.

This redesign process is part of this dissertation which also has the following goals to fulfill:

1. Define the design flow for real-time multi-processing embedded system.
2. Provide an analysis of various problems from algorithm development to system integration.
3. The discussion covers all aspects of embedded system design : Application, middleware, and service
4. Focus on steps for designing and implementation a real-time embedded system in terms of modeling (both mathematical and object modeling), analysis, and optimization of the embedded system application.

The redesigned system with all of the new added tasks will be known as a real-time road surface analyzer system which will perform different tasks depending on:

- The system to be used for collecting data regarding road surface, and then processing the raw data to determine the road surface characteristics in real-time manner, and
- The measurements obtained from a number of different sensor types including lasers, accelerometers, gyroscopes, and distance encoders.

This research involves the design, analysis, and partial implementation of a real-time embedded multi-core system for measuring, collecting, processing, and displaying pavement surface characteristics. The system obtains measurements from a number of different sensor types including lasers, accelerometers, and distance encoder. The collected data is then analyzed in order to specify various pavement surface characteristics such as transverse and longitudinal profiles, texture, as well as determining the road usability condition.

The real-time measurements will be processed using predefined as well as newly developed analysis techniques to quickly identify and quantify surface profile and texture parameters at highway speeds with minimal or no disruption to road users. Numerous pavement performance inferences can be computed concerning ride quality (roughness, smoothness), safety (friction and geometry), and durability (roughness, deformation, texture). The real-time analysis will require the use of a scalable multi-core system in order to perform the necessary computations. In Figure 1.1 road surface deficiencies and a general idea about the final product are shown.

The applications that are related to pavement surface analysis to be implemented here can be categorized as ride measurement and analysis, bump detection, 3-D surface profile and pavement texture analysis.

In order to run all of these functions in real time, the use of parallel or multi-core processing is needed as well as several hardware and software components required for the purpose of collecting and analyzing the data.

This research investigates real-time embedded multi-core systems, in order to design and implement a road surface analyzer system as an embedded application. The research plan can be summarized in the following points:

1. The appropriate tool (or tools) to model the problem under study, along with modeling the hardware and software requirements will be found.
2. Candidate processors will be tested and their performances evaluated to specify which of them supports multitasking. Then the optimal number of tasks that can be executed in parallel for every given processor will be found. It will be necessary to decide the best fitting processor for partial or full implementation for the application under study in this research.
3. The operating systems dedicated for embedded applications will be studied and one selected that can work as part of the road surface analyzer system in terms of supporting the hardware devices, having the ability to handle multitasking and parallel processing, and taking care of real-time constraints.
4. A programming language will be selected along with the libraries that handle hardware interfacing and parallel processing execution.
5. Tasks will be determined and implemented as part of the system and analyzed to define if there is any problem to be solved. It will then be necessary to specify the requirements to meet real-time execution.
6. Finally, implementation of the system and collection of data for performance analysis will be completed.

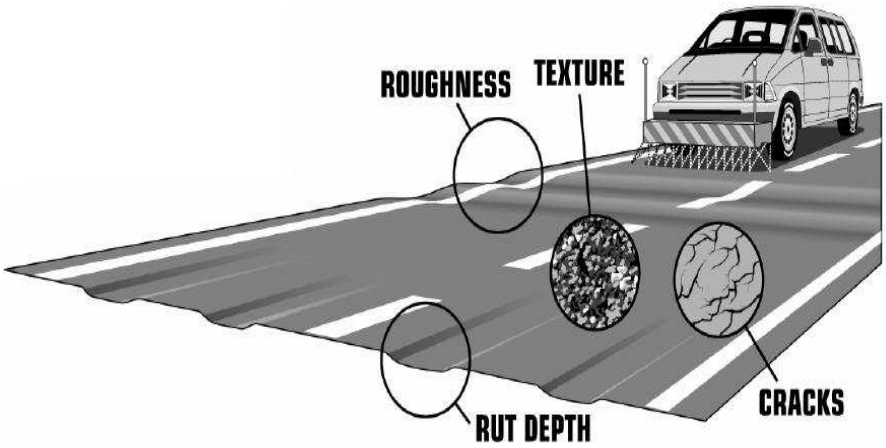


Figure 1.1 Road surface deficiencies

1.4 Organization

The remaining part of the dissertation is organized as follows. Chapter 2 introduces and compares different object modeling tools. Chapter 3 discusses and evaluates the most recently announced processors that are considered in this dissertation for the implementation of embedded systems. Chapter 4 describes the software requirements to support embedded systems starting from the operating systems dedicated to embedded applications and the programming languages and libraries that can support real time applications along with multi-threading characteristics of such systems. In Chapter 5 the road profiler system requirements are discussed in detail in terms of modeling and simulation of the main part of this system which is the quarter-car model. Also in this chapter the analytical algorithms that are used to analyze profiler data are introduced and modeled. Chapter 6 presents a detailed implementation of a profiler system and the results obtained from such a system. Finally, Chapter 7 concludes the dissertation.

CHAPTER 2

SYSTEM MODELING TECHNIQUES

2.1 Introduction

This chapter will introduce modeling methods using Unified Modeling Language (UML). UML is a standardized modeling language used for object and component modeling. UML is a general purpose modeling language that includes a set of graphical notation techniques to create abstract models of specific systems.

UML provides extensions and profiles for specific applications such as real-time systems, and hardware/ embedded applications.

In this chapter a number of well known and most common UML profiles for modeling real-time systems (both software and hardware) will be examined and compared. These are namely UML, UML-RT, SysML, SPT, and MARTE. These profiles and extensions will be compared based on attributes like hardware modeling, time representation, availability of real-time notations, and allocation.

2.2 UML Modeling Language

The OMG specification states: The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.

UML models can be customized to provide a generic extension mechanism for particular domains and platforms, this capability create what is known as UML profile. A UML

profile has the ability to define additional diagram types or extend existing diagrams with additional notations.

Extension mechanisms allow refining standard semantics in strictly additive manner, so that they can't contradict standard semantics. Profiles are defined using stereotypes, tag definitions, and constraints that are applied to specific model elements, such as Classes, Attributes, Operations, and Activities.

UML provides several diagrams used in the design procedure. The first step in modeling with UML is to provide a description of a system's behavior which is done with the aid of use case diagram. Use case describes the interaction between an actor and the system itself, represented as a sequence of simple steps. Each use case is a complete series of events, described from the point of view of the actor. Actors may be end users, other systems, or hardware devices which exist outside the system under study. Each use case is a complete series of events, described from the point of view of the actor.

Another UML diagram is the activity diagram which is used to show workflows (flowchart) in a step by step manner for the activities and actions, with support for choice, iteration and concurrency. Activity diagrams consist of initial node, activity final node, and activities. The starting point of the diagram is the initial node, and the activity final node is the ending. An activity diagram can have zero or more activity final nodes.

Another used diagram is the class diagram. The class diagram is the main building block in object oriented modeling. They are being used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed. In the conceptual design of a system a number of classes are identified and grouped together in a class diagram, which helps to determine the static relations between those objects. With detailed modeling the classes of the conceptual design are often split in a number of subclasses [11].

UML 2.0 is considered the current revision of UML. UML 2.0 represents a significant milestone in the evolution of UML to support complex systems. Some concepts from UMLRT/ ROOM, such as the notion of ports, are now standard.

The representation of the structural aspects of a system becomes clearer with the composite structure diagrams.

The capability to model dynamic and parallel behavior has drastically improved with UML 2.0, e.g.

- Sequence diagrams can have fragments. This allows the representation of loops, alternative sequences and parallel message exchanges.
- Activity diagrams are based on Petri nets and improve the representation of concurrent flows of operation.
- Timing diagrams are new and stress the importance of time when showing the interaction between objects and their change in state.

Beyond these notation enhancements, the Model Driven Architecture initiative of OMG promotes the development of platform independent (PIM) UML models which are mapped onto platform specific (PSM) UML models. This is a particular approach to model-driven development, which emphasizes the role of (executable) models. [7]

UML does not restrict UML element types to a certain diagram type. In general, every UML element may appear on almost all types of diagrams.

2.3 Modeling Hardware with UML and UML profiles

2.3.1. UML-RT

Real-time Object Oriented Modeling (ROOM) [1] is a modeling language used for modeling real-time systems. It has its own graphical notation set to model structures of real-time systems. A capsule stereotype was introduced by ROOM to represent a reactive object. A Capsule can communicate with other capsules through ports, which are boundary objects, and a protocol associated with the port. ROOM also defines a connector which connects ports to

provide transmission facility for supporting a particular protocol. ROOM is more oriented towards the actual implementation and physical design [2]. ROOM was integrated as part of UML to form what is known as UML-RT. Lack of usage and support is considered one of the limitations of the UML-RT.

Real-Time UML (RT-UML) is different than the UML-RT. RT-UML was developed and adopted and used as a preparatory module by I-Logix. It uses the basic UML notations and stereotypes to model elements in real-time and embedded domain.

2.3.2. UML Profile for Systems Modeling Language (SysML)

SysML is domain-specific Modeling language for systems engineering. SysML supports the specification, analysis, design, verification and validation of a broad range of complex systems [3]. SysML defines two types of diagrams, the Block Definition Diagram (BDD) and the Internal Block Diagram (IBD). The BDD is based on UML Class Diagrams and UML Composite Structure Diagrams. The role of a BDD is to describe the relationships among blocks, which are basic structural elements focusing on specifying hierarchies and interconnections of the system to be modeled. The SysML IBD allows the designer to refine the structural aspect of the model. The IBD is the equivalent of the composite structure in UML. SysML lacks the constructs for modeling time.

SysML extends the application of UML to systems which are not purely software based, and can in particular be applied to design heterogeneous embedded systems.

2.3.2.1 *Block Definition Diagram*

The role of a BDD is to describe the relationships among blocks, which are basic structural elements aiming at specifying hierarchies and interconnections of the system to be modeled. A block is specified by its *parts*, and *flow ports*.

Parts represent the physical components of the block while flow ports represent the interfaces of the block, through which the block communicates with other blocks.

2.3.2.2 Internal Block Diagram

The SysML IBD allows the designer to refine the structural aspect of the model. The IBD is the equivalent of the composite structure in UML. In the IBD, parts are the basic elements of the diagram and they are assembled to define how they collaborate to realize the behavior of the block. A part represents the usage of the corresponding block. The most important aspect of the IBD is allowing the designer to refine the definition of the interaction between the usages of blocks by defining flow ports as follows:

• Ports are parts available for connection from the outside of the owning block;

• Ports are typed by interfaces or blocks that define what can be exchanged through them;

• Ports are connected using connectors that represent the use of an association in the IBD.

2.3.2.3 SysML ports

Two types of ports are available in SysML:

• *Standard ports* handling requests and invocations of services with other blocks (basically the same concept as in UML2.0);

• *Flow ports* which let blocks exchange flows of information.

Flow ports specify the interaction points among blocks and parts supporting the integration of behavior and structure. For standard ports, an interface class is used to list the services offered by the block. For flow ports, a flow specification is created to list the type of data that can flow through the port.

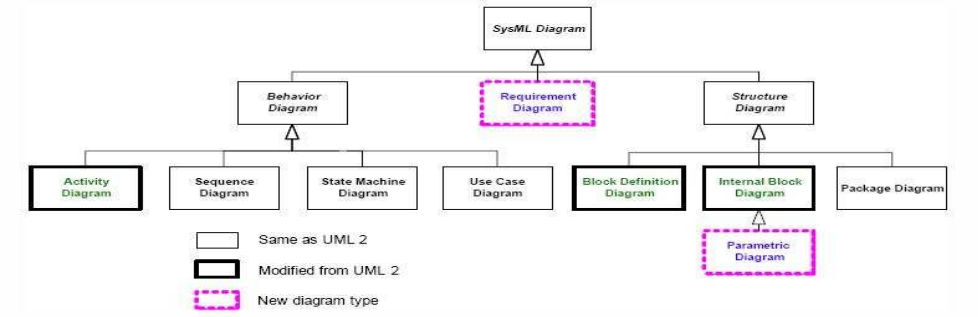


Figure 2.1 SysML Architecture

2.3.3. UML Profile for Scheduling, Performance and Time (SPT)

SPT defines a set of concepts useful for modeling real-time systems. Its purpose is to integrate notation used by existing real-time analysis techniques into UML in order to:

- Enable the construction of models that could be used to make quantitative predictions regarding these characteristics.
- Facilitate communication of design intent between developers in a standard way.
- Enable inter-operability between various analysis and design tools.

Thus, the SPT is defined to offer a common framework for real-time modeling that unifies the diversity of techniques, terminologies and notations existing in the real-time software community, while still leaving space for different kinds of specifications. In its present form, the main focus of SPT is on time and time-related concepts: performance, timelines, schedulability, etc.

SPT offers a terminology for modeling real-time systems: it defines a set of concepts - aiming to fit standard real-time modeling techniques - and some relationships between these concepts as allowed by the meta-modeling technique used for the definition of the SPT.

The use of SPT is justified because UML is lacking in some key areas that are of particular concern to real-time system designers and developers. In particular, the lack of a quantifiable notion of time and resources was an impediment to its broader use in the real-time and embedded domain. It was discovered that UML had all the requisite mechanisms for addressing these issues, in particular through its extensibility faculties [9]. SPT is a standard way of using these capabilities to represent concepts and practices from the real-time domain.

One of the main guiding principles is that, as much as possible, modelers should not be hindered in the way they use UML to represent their systems just for the purpose of model analysis. That is, rather than enforcing a specific approach or modeling style for real-time systems, the profile should allow modelers to choose the style and modeling constructs that they feel are the best fit for their needs of the moment.

2.3.4. Modeling and Analysis of Real-time and Embedded (MARTE)

MARTE [4] is UML profile adopted by OMG in order to extend the capacities of UML for real-time modeling in embedded systems. Not only for the modeling and analysis, MARTE also provides support for specification, design, and verification/ validation stages. This new profile is intended to replace the existing UML Profile for SPT. [5]

Because SPT's constructs were considered too abstract and hard to apply, and for the requirement of aligning SPT profile UML2.0, there was a need for upgrading or creating new profile.

MARTE profile is an evolution of the SPT profile with the purpose of upgrading this profile to UML2. It is made of various packages: namely MARTE foundations, MARTE design model, MARTE analysis model and MARTE annexes. The profile is intended to be a fundamental tool in the design of real time systems. Both modeling and analyzing concerns are tackled leading to a complete instrument to improve the design phase. Within MARTE, the Software Resource modeling (SRM) framework provides modeling artifacts to describe software execution platform modeling. The SRM profile provides a broad range of modeling capabilities covering main multitasking framework such as dedicated real-time language libraries and real-time operating systems.

Besides software resources, MARTE allows us to model hardware resources. Due to its general purpose, UML lacks certain key native artifacts for describing concrete and precise hardware RTE execution platform. The UML profile for MARTE fills this lack with two sub-profiles: a generic resource modeling (GRM) profile and a hardware resource modeling profile (HRM). Both can be used to model hardware platform.

The HRM is composed of two views, a logical view that classifies hardware resources depending on their functional properties, and a physical view that concentrates on their physical nature. Both are specializations of the general model. The logical and physical views are

complementary. They provide two different abstractions of hardware which could be simply merged.

In MARTE there are two kinds of time: the logical time and the chronometric time. The chronometric time concerns with the time cardinality alone, whereas the logical time concerns the ordering and organization of events which can again be synchronous or asynchronous. The specialization of the logical time model for synchronous events is independently known as the synchronous time model. For extreme real time applications the simultaneity of events occurrence is taken care by the synchronous time model. [8]

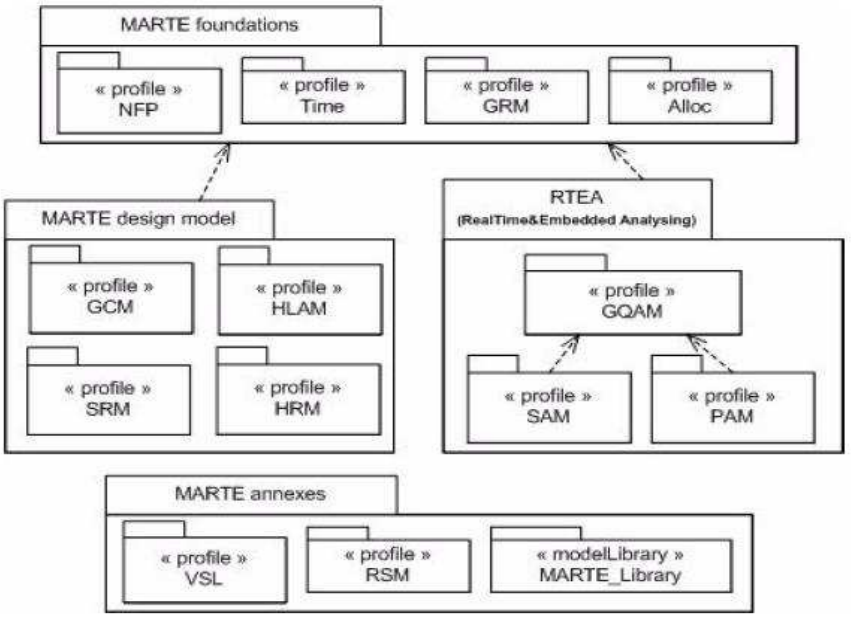


Figure 2.2 MARTE Architecture

Once the application and the hardware architecture have been modeled, one should specify how the applications will be executed on the execution platform. The mapping of application tasks onto the adapted execution platform is an important point of real-time embedded system design. [10]

In the MARTE profile, the mechanism to specify the mapping is called allocation: a MARTE allocation is an association between a MARTE application and a MARTE execution

platform. The set of all the allocations of the model defines the mapping. The main concept of allocation is « Allocate »; it is used for associating elements from a logical context, application model elements, to named elements described in a more physical context, execution platform model elements.

2.4 Real-time Modeling Using UML and UML Profiles

Common UML and/ or UML profiles modeling practices suggests that

- Use of UML profiles only when standard UML cannot perform the task.
- Use SysML diagrams for general modeling of the system.
- Use MARTE for modeling the system's details.

Use MARTE specifically to model the hardware system with all of its details along with modeling the software methods which are part of the application.

2.4.1. Concurrent (Parallel) Operations Representation

The concurrent operations are represented in the activity diagram with the aid of fork, join states. The fork pseudo-state is a connector that branches a single input transition into multiple outgoing transitions to different states that will be activated concurrently. While the join pseudo-state joins together multiple incoming transitions into a single transition. Once the data collection is over, and the system is ready to stop working, all the data (raw and calculated) will be saved in output files (database) for offline analysis and archiving process.

When using state machine diagrams, UML models parallelism in two ways. First, all objects are considered to be parallel entities. Second, a single object entity exhibits itself a concurrent behavior. This means that the object's state-machine is specified as a set of concurrent components.

CHAPTER 3

EVALUATION OF EMBEDDED PROCESSORS

3.1 Evaluating General Purpose Processors for Embedded Systems

3.1.1. Single and Multi-Core Processors

The following subsections will briefly discuss the architecture of all the processors considered for evaluation in this research, all of which are developed and manufactured by Intel as general purpose processors. Those processors can be used in different types of systems, desktops, laptops, and embedded systems. The evaluated processors support different types of the latest multiprocessing technologies. Recently, multiprocessing is implemented using either chip multiprocessing (CMP) or Simultaneous multithreading (SMT).

CMPs also known as multi-core processors are implemented by integrating two or more independent processors (cores) on a single die (or chip).

SMT, on the other hand, is a technique for improving CPU performance. The instructions from two threads are interleaved in the CPU pipeline. SMT duplicates some circuits of the processor including the some of the pipeline stages but not duplicating the main execution resources. Intel implementation of the SMT is known as Hyper-Threading (HT).

3.1.2. ATOM Processor

The Atom is a low-power processor from Intel. Power consumption for all Atom processors ranges from 2- 8W. The Atom processors are either single or dual core processors that support the hyper threading technology. Atom processors are designed with two levels of cache. Level 1 cache is of size 32 KB instruction cache, 24 KB data cache data cache per core where smaller L1 data cache implemented to minimize power consumption. The size of level 2 cache is 512KB for single core or 1 MB (2x512 KB) for the dual core Atom processors. Atom processor is designed with different micro architecture from other Intel processors. The Atom

micro architecture does not support out-of-order execution to minimize power consumption since the components in charge of issuing and controlling microinstructions execution could be removed.

3.1.3. EP80579 System on Chip

The Intel EP80579 (code name : Tolapai) [20] is a system-on-a-chip (SoC) embedded processor which includes an Intel architecture complex based on the Intel Pentium M processor, integrated memory controller hub, integrated I/O controller hub and flexible integrated I/O support with three Ethernet connections, two Controller Area Network (CAN) interfaces and a local expansion bus interface. The design also includes PCI Express, High Speed Serial1 (HSS) ports for TDM or analog voice connectivity, security accelerators for bulk encryption, hashing and public/private key generation.

The Intel QuickAssist Technology initiative consists of a family of interrelated Intel and industry standard technologies that simplify the use and deployment of accelerators on Intel platforms. The integrated accelerators in this processor support Intel QuickAssist Technology through software packages provided by Intel. These software packages provide the library structures to integrate security and/or VoIP functionality into the application, completely adjunct to the Intel architecture complex, freeing up CPU cycles to support additional features and capabilities. This provides the efficiency of customized hardware with the flexibility to design diverse applications with one platform. Figure 3.1 shows a block diagram of the Tolapai SoC

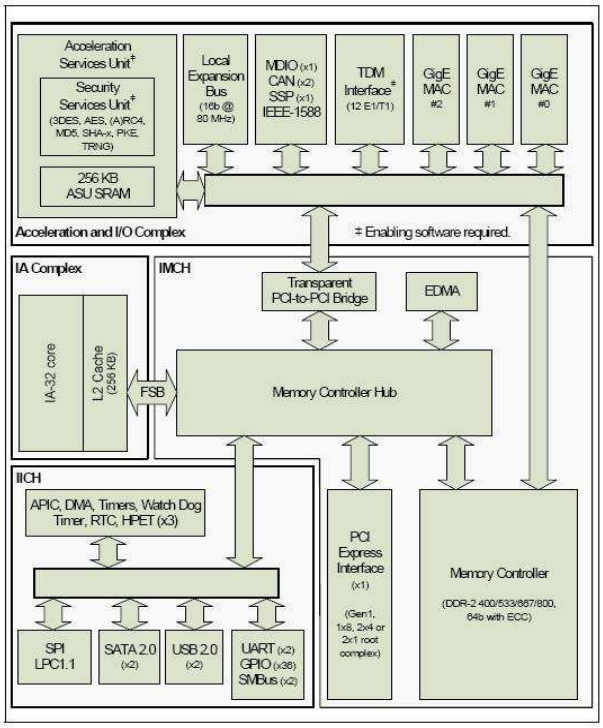


Figure 3.1 Block Diagram of the Tolapai Embedded Processor [20]

3.1.4 Core 2 Quad

The core 2 quad processor takes two dual core processors (core 2 duo) and combines them into a single package. Each of the Core 2 duo processors features 4MB of Advanced Smart Cache, which is shared between the two cores. Combining two core 2 dual in a single chip allows the new core 2 quad processors to have a total of 8MB L2 cache, but without the ability of sharing the entire L2 cache among all the four cores. Instead it acts like dual Core 2 processors, each sharing 4MB of L2 cache. Each core has a level 1 cache of size 32 KB instruction and 32 KB data. This affects the entire processor, since without any shared resources between the dual processor die, duplication between the two distinct processors may occur. Finally, hyper-threading is not implemented nor supported by the core 2 processors in general and the core 2 quad in particular. The Core 2 Quad Q6600 processor features an Intel

quad core running at 2.4 GHz with an 8MB (2x4MB) of combined L2 cache. Figure 3.2 illustrates the core 2 quad architecture.

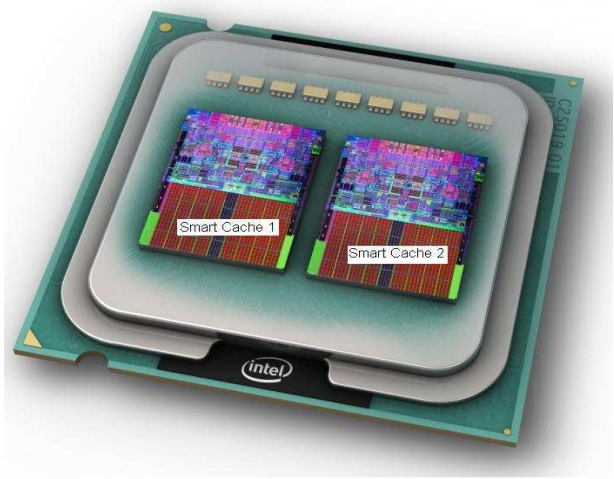


Figure 3.2 Core 2 Quad Illustration [12]

3.1.5 Core i7

The core i5 and i7 processors are the newest multi-core processors from Intel, and are successor to the Intel Core 2 family. The core i7 processors are quad core processors that support the hyper threading technology, and come with many new features to enhance the processor's performance over the core 2 processors. The core i7 has an on-chip memory controller which means that the memory is directly connected to the processor. This memory controller is a triple-channel controller that supports DDR3 memory only. Also the front side bus has been replaced by the Intel QuickPath Interconnect (QPI) interface. The QPI is a packet-based point-to-point connection between the processor and the I/O chipset. Core i7 designed is with three levels of cache. Level 1 cache is of size 32 KB instruction and 32 KB data cache per core, level 2 size is 256 KB combined instruction and data per core, and level 3 cache is an 8 MB on-chip smart cache shared among all four cores. Figure 3.3 represents the core i7 architecture.

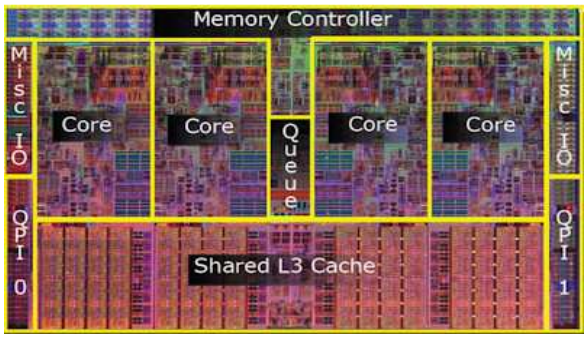


Figure 3.3 Core i7 Architecture [13]

3.1.6 Xeon Quad core

According to Intel [10], the Intel Xeon processor and Intel 5100 memory controller hub chipset development kit provides compelling performance-per-watt advantages for thermally constrained applications in the storage, communications and embedded market segments. This kit supports two Xeon processors. The processor used here is the Xeon L5408 which is one of the low-power quad core Xeon processors, thermal power for L5408 is 40 W in comparison to 80- 120 W for regular Xeon quad core processors. It runs at clock rate of 2.13 GHz with level 1 cache of size 32 KB instruction and 32 KB data per core, 12 MB of L2 cache (each 2 cores share 6 MB), and 1066 MHz FSB speed. Figure 3.4 is an illustration for the kit used.

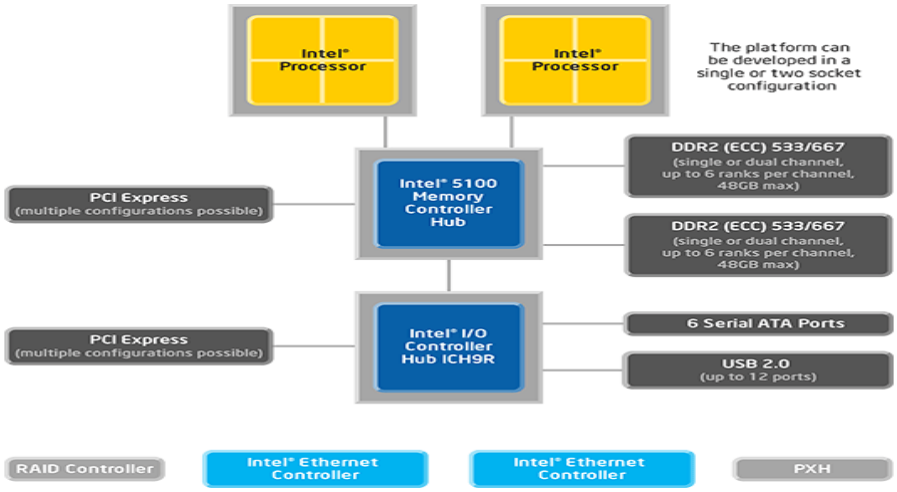


Figure 3.4 Intel Xeon Processor and the Intel 5100 Memory Controller Hub Block Diagram

3.2 Benchmark Suits

Performance evaluation of computer systems in general and processors in particular becomes more difficult with the advancement of these systems. For that reason, tests were developed to analyze and compare between different systems. Benchmarks are designed to mimic a specific workload to test the whole computer system or a certain part of it such as the processor, memory, I/O devices, or network communications. Benchmarks perform tests by either creating special programs that impose the workload on the component or by using a real-world application then run it on the system under test. In our work we used three different types of benchmark suites to test the processor's computational speed as well as the memory performance. The following subsections discuss these benchmark suites.

3.2.1. Lmbench

Lmbench is a suite of simple, portable micro-benchmarks for UNIX. Lmbench measures two key features memory bandwidth and latency. It measures systems ability to transfer data between processor, cache, memory, network, and desk [17].

Lmbench contains a large number of micro-benchmarks that measure various aspects of hardware and operating system performance. It generally reports the median result for 11 measurements. The following table lists the memory operations carried by this benchmark.

Table 3.1 Memory operations from Lmbench [14]

Operation	Description
Libc bcopy unaligned	Measuring how fast the processor can copy data blocks when data segments are not aligned with pages using a function bcopy()
Libc bcopy aligned	Measuring how fast the processor can copy data blocks when data segments are aligned with pages using a function bcopy()
Memory bzero	Measuring how fast the processor can reset memory blocks using a function bzero()
Unrolled bcopy unaligned	Measuring how fast the system can copy data blocks without using bcopy(), when data segments are not aligned with pages
Memory read	Measuring the time to read every 4 byte word from memory (stride 32 bytes)
Memory write	Measuring the time to write every 4 byte word to memory (stride 32 bytes)

3.2.2. NBP Benchmark

NPB benchmarks [15] target performance evaluation of highly parallel computers. NPB stands for NAS (Numerical Aerodynamic Simulation) Parallel Benchmark, which is developed and maintained by the NASA Ames Research Center. The NPB mimics the computation and data movement characteristics of large scale computational fluid dynamics (CFD) applications.

These benchmarks are written in FORTRAN and C programming languages with the aid of OpenMP to achieve parallelism.

The NPB set consists of several benchmarks shown in Table 3.2. Those benchmarks used to simulate some of the following problems: [22]- [24]

EP: An embarrassingly parallel kernel. It provides an estimate of the upper achievable limits for floating point performance. Embarrassingly parallel problem is one for which little or no effort is required to separate the problem into a number of parallel tasks. This is often the case where there exists no dependency between those parallel tasks.

MG: A simplified multi grid kernel. It requires highly structured long distance communication and tests both short and long distance data communication.

CG: A conjugate gradient method is used to compute an approximation to the smallest eigenvalue of a large, sparse, symmetric positive definite matrix. This kernel is typical of unstructured grid computations in that it tests irregular long distance communication, employing unstructured matrix vector multiplication.

FT: A 3D partial differential equation solution using FFTs. This kernel performs the essence of many spectral codes. It is a rigorous test of long-distance communication performance.

IS: A large integer sort kernel that performs a sorting operation that is important in particle method codes. It tests both integer computation speed and communication performance.

BT, SP, and LU benchmarks solve a synthetic system of nonlinear Partial differential equations (PDEs) using three different algorithms involving block tri-diagonal, scalar pentadiagonal and symmetric successive over-relaxation (SSOR) solver kernels, respectively.

Data Cube (DC) benchmark takes a synthetic dataset described by a small number of parameters and generates multiple views of this set. Informally, it can be classified as multidimensional sorting. Multiple processors can work in parallel to measure combined performance of multiple I/O systems attached to a machine. Furthermore, the parameters of the input dataset can be chosen to saturate I/O systems of the largest existing machines, so that multiple hosts may be efficiently used to reduce the benchmark turn-around time.

The DC benchmark performs a data-intensive operation known in data mining as the Data Cube Operator (DCO).

Table 3.2 NPB 3.3 Benchmarks [15]

Benchmark	Name derived from
MG	Multi-Grid
CG	Conjugate Gradient
FT	Fast Fourier Transform
IS	Integer Sort
EP	Embarrassingly Parallel
BT	Block Tri-diagonal
SP	Scalar Penta-diagonal
LU	Lower-Upper symmetric Gauss-Seidel
UA	Unstructured Adaptive
DC	Data Cube operator
DT	Data Traffic

Problems solved by the NPB benchmarks are classified in different problem classes according to the problem size. The classes, according to the problem size they represent, are S, W, A, B, C, D, and E; where S is the smallest and E is the largest.

3.2.3. STREAM/ STREAM2

The STREAM/ STREAM2 benchmarks [16] measure memory bandwidth and latency based on the most common functions as stated in Table 3.Table 3.3.

STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth and the corresponding computation rate for simple vector kernels. It is intended to characterize the behavior of a system for applications that are limited in performance by the memory bandwidth of the system, rather than by the computational performance of the CPU. [18]

STREAM2 is based on the same ideas as STREAM, but uses a different set of vector functions. It is an attempt to extend the functionality of the STREAM benchmark in two important ways (1) measure the sustained bandwidth at all levels of the cache hierarchy, and (2) more clearly expose the performance differences between reads and writes.

Table 3.3 STREAM/STREAM2 functions

Function	Code	Benchmark
COPY	$a(i) = b(i)$	STREAM
SCALE	$a(i) = q * b(i)$	STREAM
ADD	$a(i) = b(i) + c(i)$	STREAM
TRIAD	$a(i) = b(i) + q * c(i)$	STREAM
FILL	$a(i) = q$	STREAM2
COPY	$a(i) = b(i)$	STREAM2
DAXPY	$a(i) = a(i) + q * b(i)$	STREAM2
SUM	$sum = sum + a(i)$	STREAM2

Where a, b, and c are vectors, i is vector index, q is constant, and sum is variable.

See [16] for more details about STREAM/ STREAM2 benchmarks.

3.2.4. Benchmark Results

All benchmarks were executed on different systems to support each of the processors' requirements. For all tested processors, the number of copies for each of the benchmarks was varied between single copy and up to 16 copies (threads) to test the performance of the processors with multithreaded applications except for the EP80579 were the number of threads varied from one up to four. The operating system used to evaluate The ATOM (single and dual core), the core 2 quad, and the core i7 processors was Ubuntu 9.04 Linux with 2.6.28-16-

generic kernel. While Red Hat 5.3 was the operating system used to evaluate the EP80579 SoC and the quad core Xeon processor.

For all benchmarks, the system specifications were as follows:

1. For the EP80579, the Intel EP80579 development kit system running at 1.2 GHz, with 1 GB of RAM.
2. For The single and dual core ATOM processors (ATOM 230 and 330), both the processors run at 1.6 GHz clock speed, with 512 KB, and 1 MB L2 cache respectively, and 2 GB DDR2 RAM.
3. The core 2 Quad Q6600 runs at 2.4 GHz clock speed with an 8MB of combined L2 cache and 4 GB of DDR2 RAM.
4. For quad core Xeon, the Intel Xeon processor and Intel 5100 memory controller hub chipset development kit was used. The kit comes with dual quad core Xeon L5408 processors; each of them runs at 2.13 GHz clock speed, having 12 MB L2 cache, and 1 GB of DDR2 RAM.
5. Finally, the Intel core i7- 920 based system; which runs at 2.66 GHz clock speed, with 8 MB L3 cache, and 6 GB DDR3 RAM.

3.2.4.1 Lmbench Results

Figures 3.5 through 3.13 represent the memory bandwidth different memory tests carried by the Lmbench benchmark. The memory bandwidth results shown in the figures were obtained using one, two, and four copies of the benchmark in case of the EP80579 SoC, and using one, two, four, and eight copies of the benchmark for the rest of processors.

Table 3.4 lists the memory bandwidth from single copy run for all processors when using a 512 Bytes array size.

The EP80579 memory bandwidth (Figure 3.5) has three distinctive regions; the first region that has the highest memory bandwidth is obtained when the memory is accessed array size less than or equal to 32 KB which is the size of the L1 data cache, for the memory read and

write operations the bandwidth is constant in this region while for the other memory operations (all kind of copy operations) the bandwidth increases by increasing the array size until reaching the 16 KB then starts dropping. The bezero operation increases memory bandwidth until it reaches its peak with 32 KB memory array size then it drops. For all memory access operations tested by the Lmbench, when accessing memory blocks with sizes larger than 32 KB and up to 512 KB the bandwidth drops linearly by about 50% for each size increment until it reaches the lowest bandwidth with the 512 KB. The third region is when accessing memory blocks larger than the L2 cache size (512 KB) in this case the bandwidth is constant and in general it represents the main memory bandwidth since with memory blocks larger than 512 KB the data resides in the main memory. The number of copies used didn't affect the memory bandwidth results.

In case of the ATOM processor both single (Figures 3.6 and 3.7) and dual core (Figures 3.8 and 3.9), the memory bandwidth drops when the array size larger than 16 KB due to the sizes of their L1 cache per core (Lmbench does not test memory bandwidth for arrays of size 24KB). Increasing the array size from 256KB to 512 KB causes a drop in memory bandwidth by around 50%, increasing the size of the array even larger than 512 KB (the size of L2 cache) drops the memory bandwidth by another 50%. For the single core ATOM running more copies of the benchmark didn't affect the memory bandwidth results. While for the dual core ATOM the memory bandwidth doubles when running more than two, four, or eight copies in comparison to the results from the single copy execution.

The memory bandwidth delivered from the Xeon based system doubles every time the copies doubled even when using eight copies, taking the advantage of using dual processor system. It can also be noticed that both provide the same performance when running up to four copies of the benchmark with the advantage of the core 2 quad over the Xeon quad core processor. On other hand, the performance obtained when running the bcopy function (either for the aligned or unaligned) with up to four copies, the Xeon based system provides

performance about 60% better than the core 2 quad system. Also, for Xeon quad core, there are two noticeable drops on the memory bandwidth, one when array sizes exceed 32 KB (L1 cache size), and the other one when the size is larger than 4 MB due to L2 cache size. For instance, for the memory bzero bandwidth results from Figure 4.b when using arrays larger than 32 KB the memory bandwidth drops approximately by 43%, but when increasing array size to 8MB the memory bandwidth drops by more than half of its bandwidth at 4MB.

The core i7 processor (with or without HT) outperforms the core 2 quad at least by three times the amount of data transferred during the benchmark runs. Also, it is easily noticed that the memory bandwidth doubles when running two copies in comparison to single copy runs.

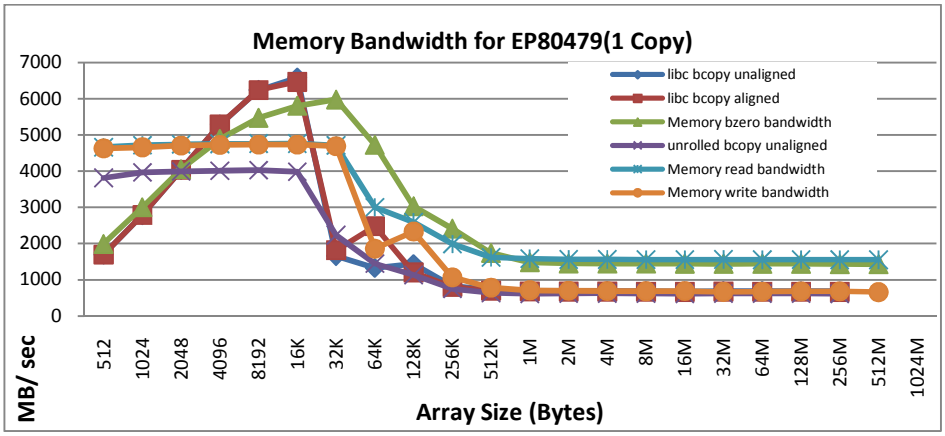
For core 2 quad processor, In general there the bandwidth drops when the array size larger than 32 KB due to the sizes of their L1 cache per core, also there is a second drop in the bandwidth when using array sizes larger than 2 MB because of the L2 cache, since each two cores share a 4 MB of L2 cache. For instance, for the memory bzero bandwidth results from figure 4 when using arrays larger than 32 KB the memory bandwidth drops by about 30%, but when increasing array size from 2MB to 4MB it drops about 42%, then increasing it above 8MB it drops by about 70%.

On the other hand, core i7, there are two noticeable drops on the memory bandwidth, one when array sizes exceed 32 KB (L1 cache size), and the other one when the size is larger than 2 MB (due to L3.)There is a third slight drop when exceeding the 128 KB size. For instance, for the memory bzero bandwidth results from figure 6 when using arrays larger than 32 KB the memory bandwidth drops approximately by 43%, but when increasing array size from 2MB to 4MB it drops by 23%, then increasing it above 8MB it drops by 30%. When disabling the hyper threading, the memory bzero bandwidth results from figure 8 when using arrays larger than 32 KB the memory bandwidth drops by about 50%, but when increasing array size from 2MB to 4MB it drops by 24%, then increasing it above 8MB it drops by about 29%.

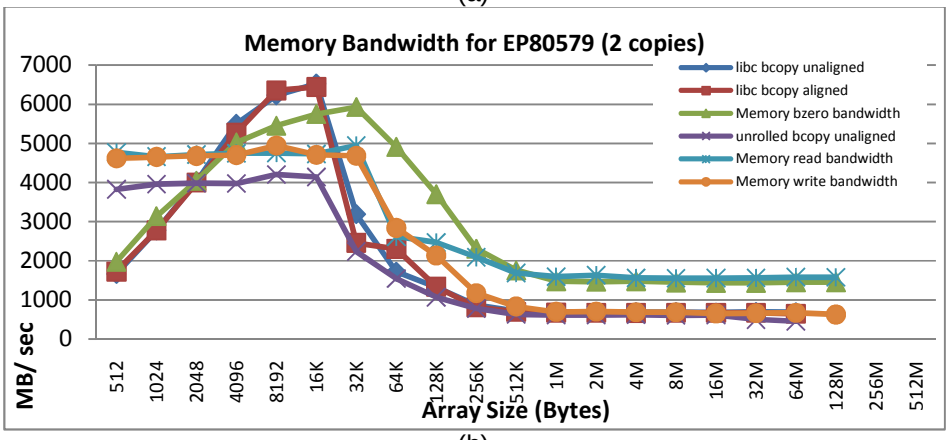
The Hyper-threading technology implemented in the ATOM and the core i7 processors have negligible effect on the memory bandwidth.

Table 3.4 Memory Bandwidth (in MB/ sec) results from Single Copy Run with Array Size 512 Bytes

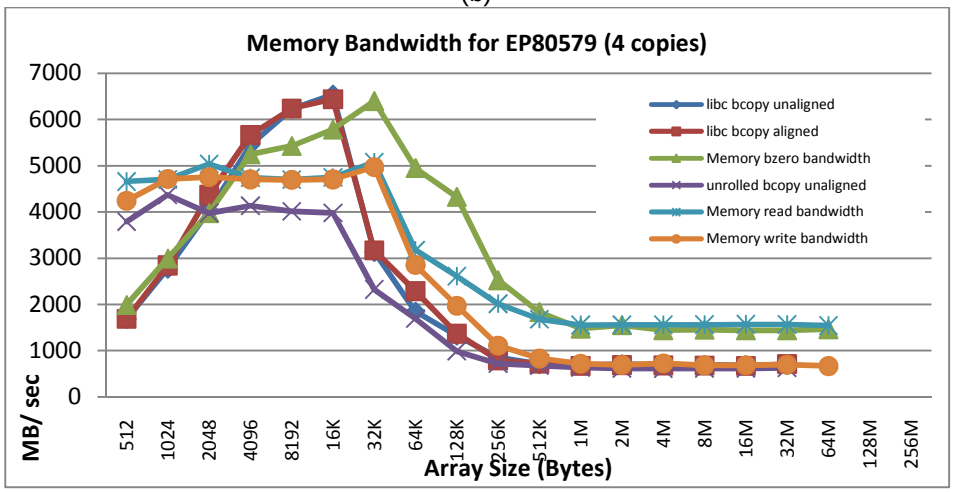
Processor	libc bcopy unaligned	libc bcopy aligned	Memory bzero bandwidth	unrolled bcopy unaligned	Memory read bandwidth	Memory write bandwidth
EP80579	1693.33	1693.33	1991.33	3815.32	4660.86	4630.21
Dual Core ATOM 330 no HT	2223	2220.31	2903.66	3114.52	6045.94	5164.86
Dual Core ATOM 330 with HT	2223.63	2223.71	2893.14	3113.39	6042.92	5159.05
Single Core ATOM 230 no HT	2224.34	2224.35	2903.97	3115.65	6047.04	5164.49
Single Core ATOM 230with HT	2217.5	2220.31	2898.92	3110	6035.06	5159.54
Core 2 Quad	3726.73	3726.39	8560.71	6227.04	9346.22	6148.52
Xeon	4387.86	5853.53	6002.9	8237.88	8311.66	5961.66
Core i7 with HT	6108.65	6099.79	14193.94	10860.21	7339.44	11643.72
Core i7 no HT	6108.73	6108.66	14196.62	10878	7340.32	11645.85



(a)

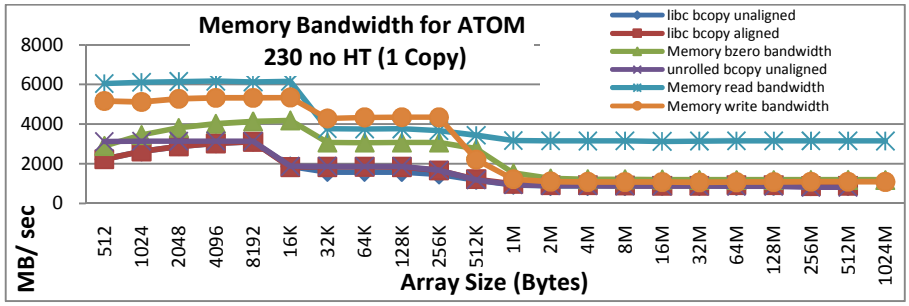


(b)

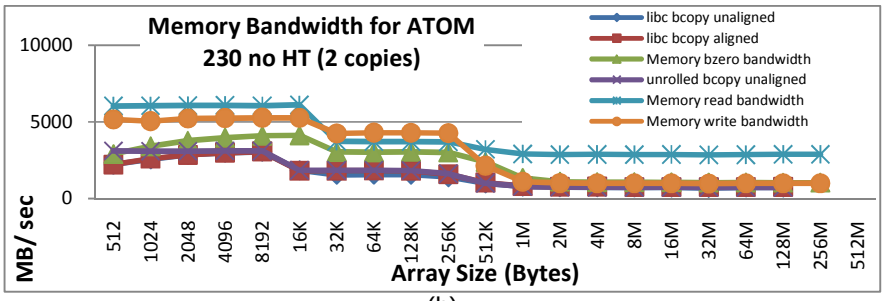


(c)

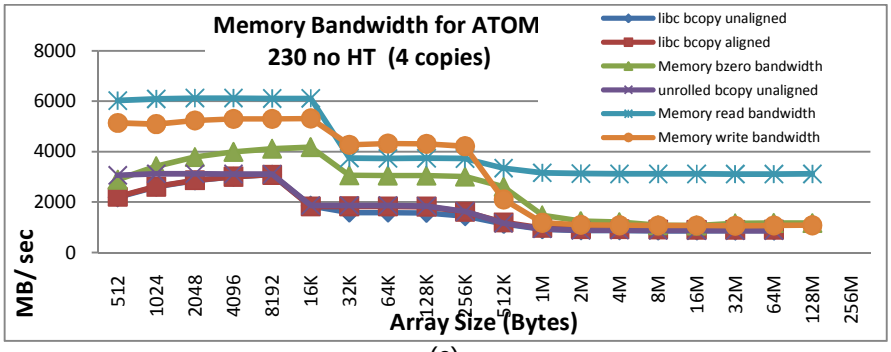
Figure 3.5 Memory Bandwidth for EP80579 SoC using (a) 1 Copy, (b) 2 Copies, and (c) 4 Copies



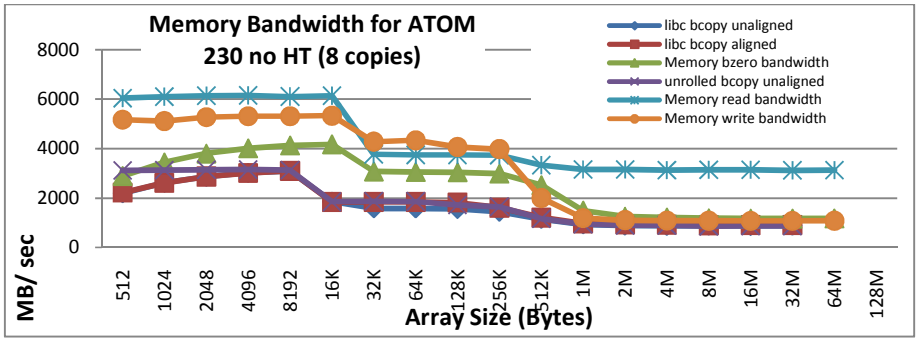
(a)



(b)

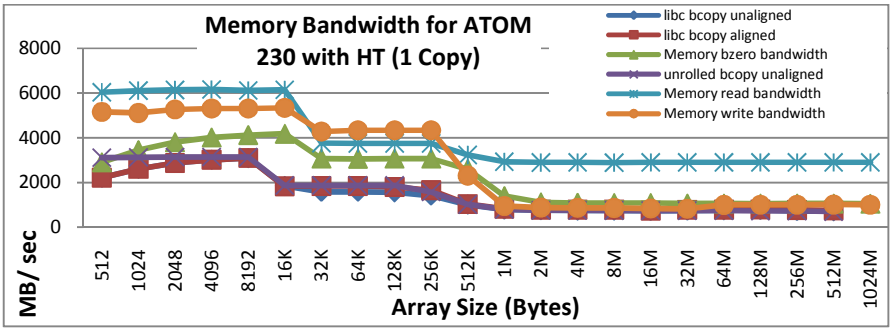


(c)

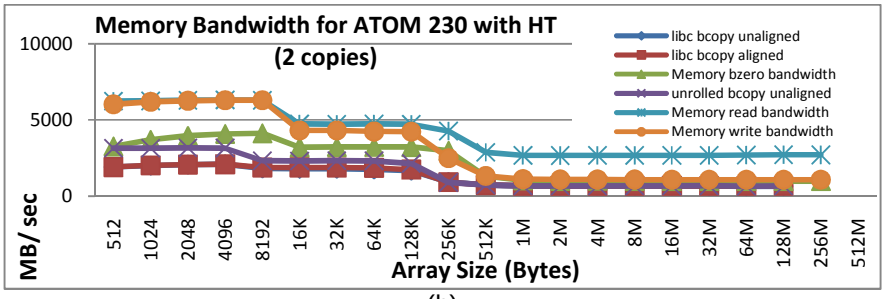


(d)

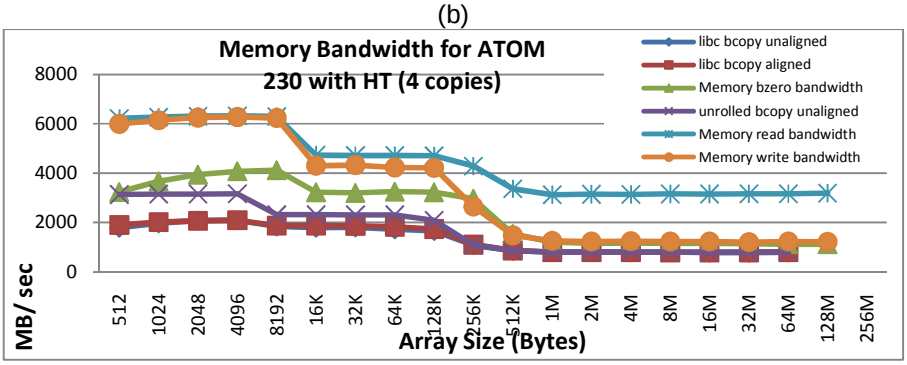
Figure 3.6 Memory Bandwidth Results for ATOM 230 (HT Disabled) using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies



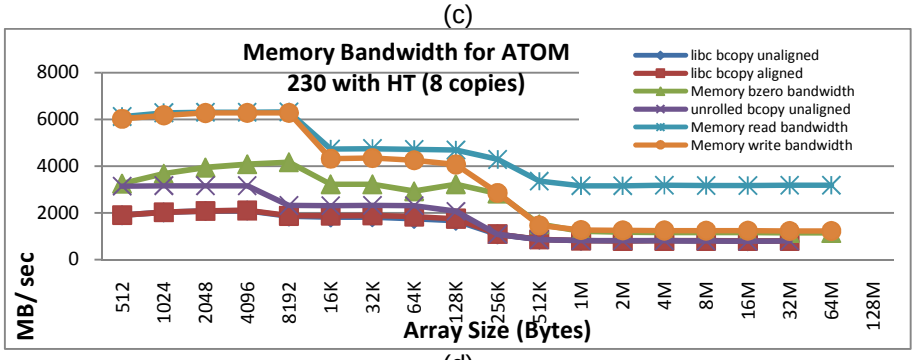
(a)



(b)

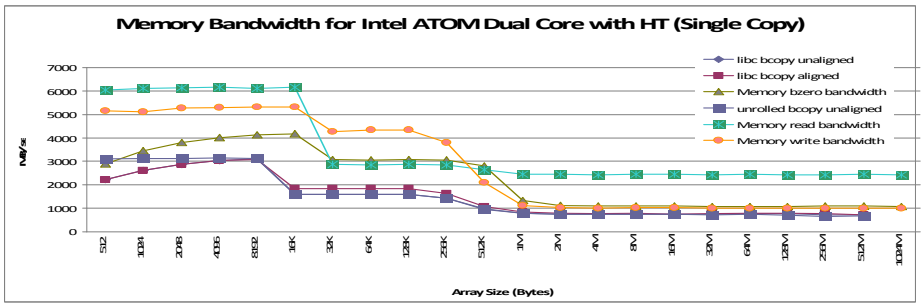


(c)

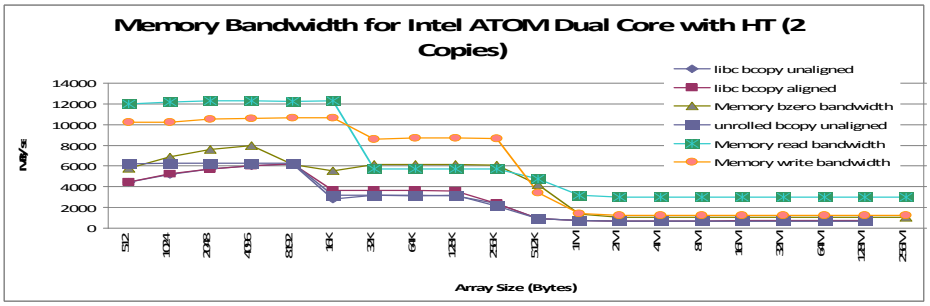


(d)

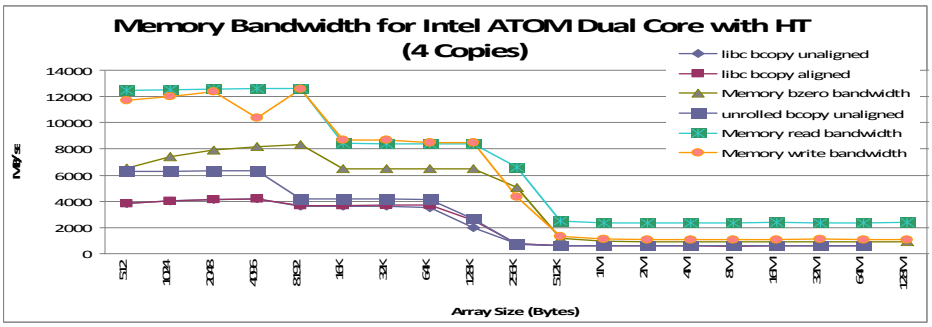
Figure 3.7 Memory Bandwidth Results for ATOM 230 (HT Enabled) using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies



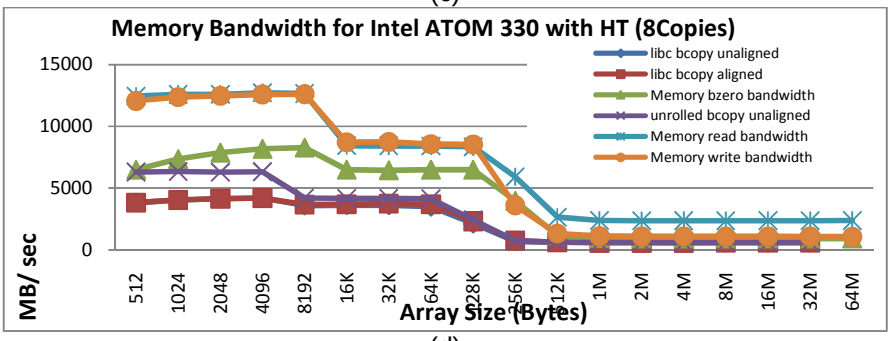
(a)



(b)

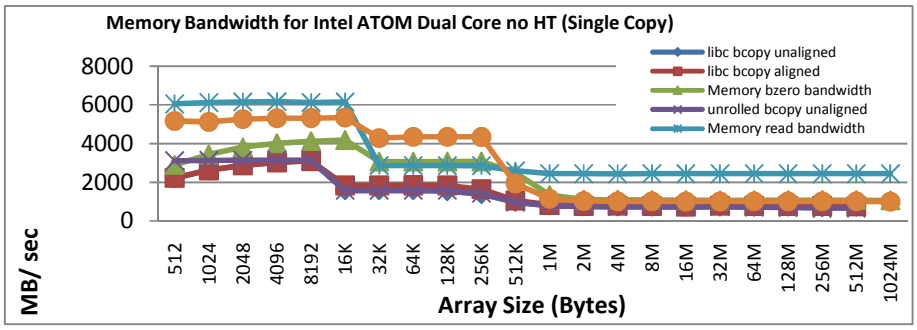


(c)

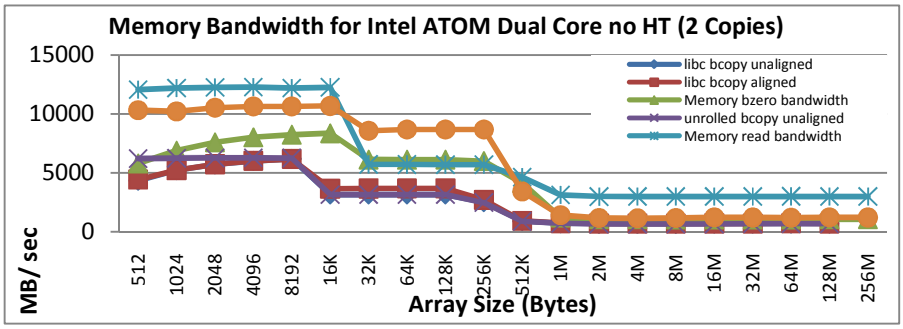


(d)

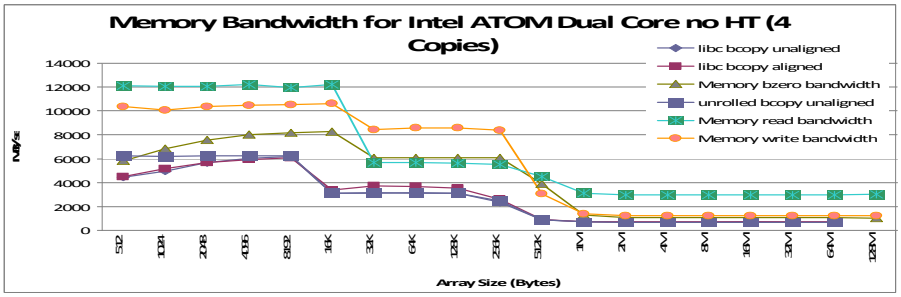
Figure 3.8 Memory Bandwidth Results for ATOM 330 (HT Enabled) using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies



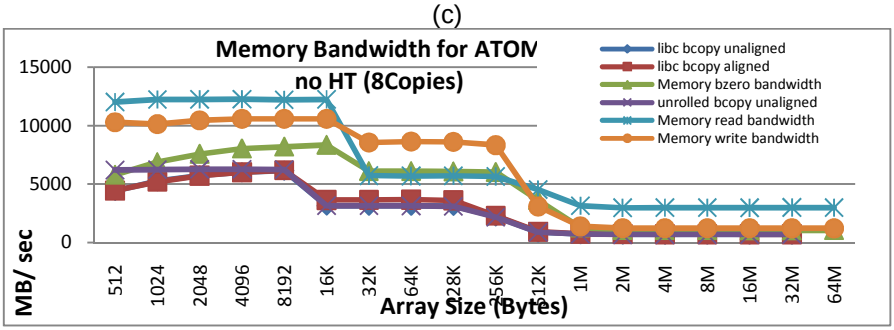
(a)



(b)

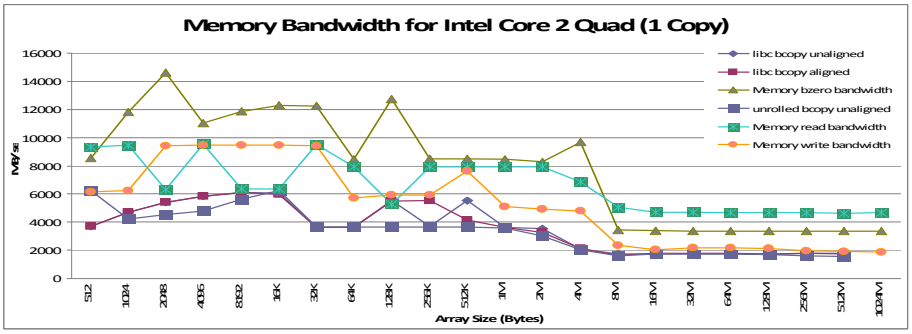


(c)

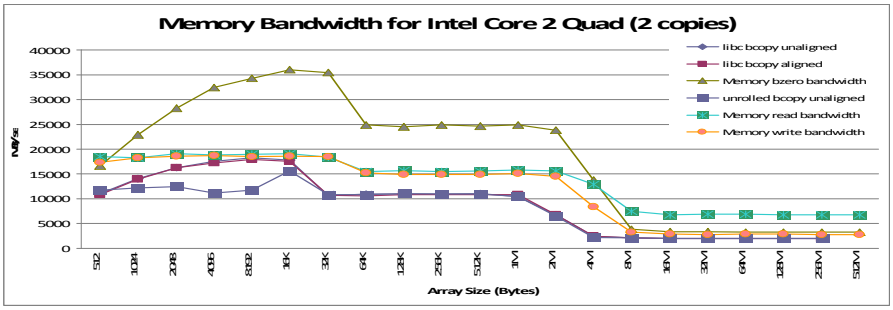


(d)

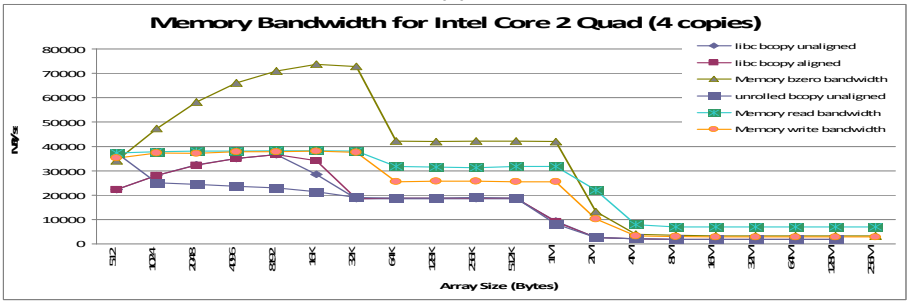
Figure 3.9 Memory Bandwidth Results for ATOM 330 (HT Disabled) using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies



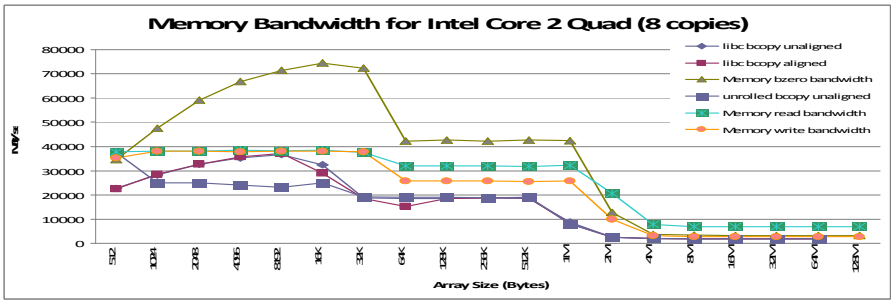
(a)



(b)

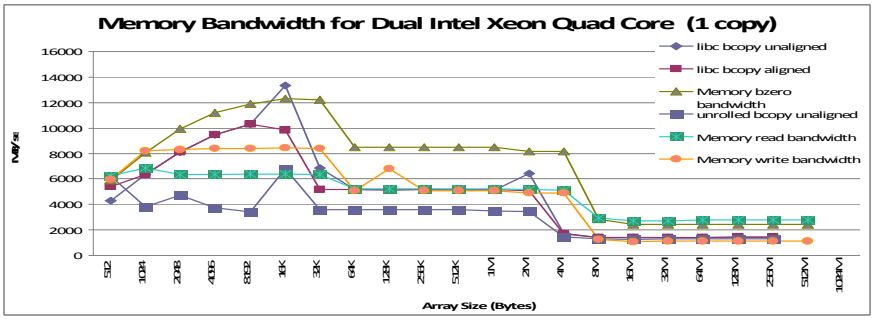


(c)

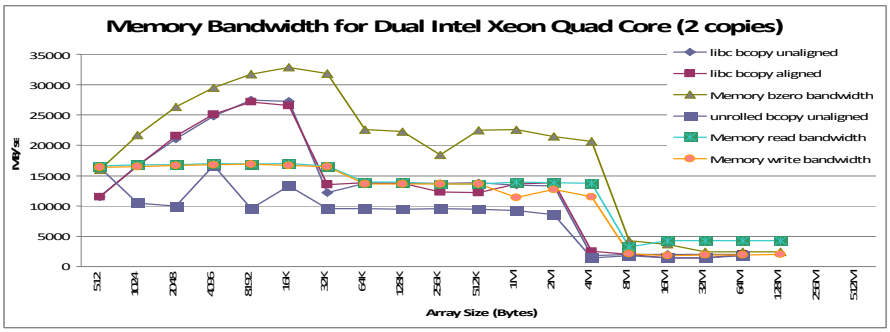


(d)

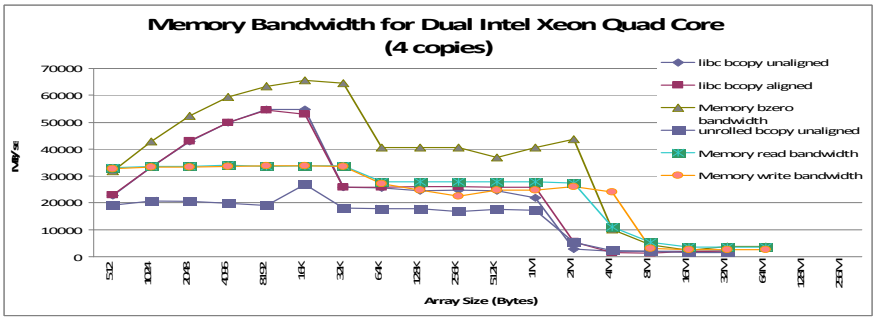
Figure 3.10 Memory Bandwidth Results for Core 2 Quad using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies



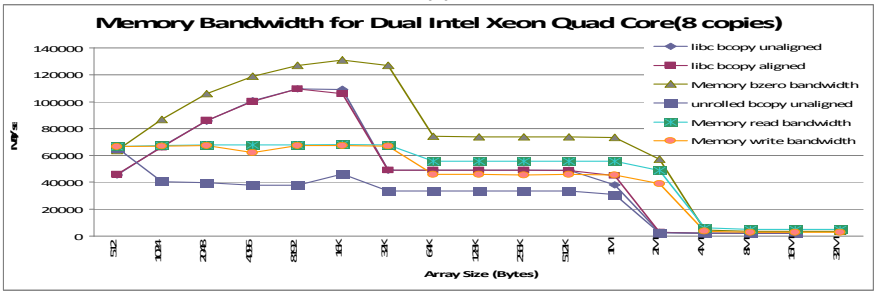
(a)



(b)

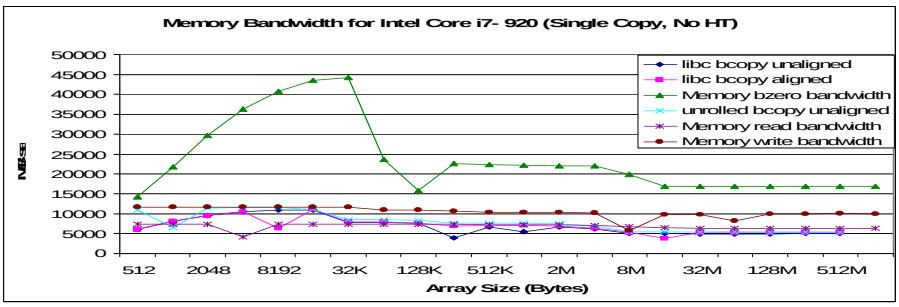


(c)

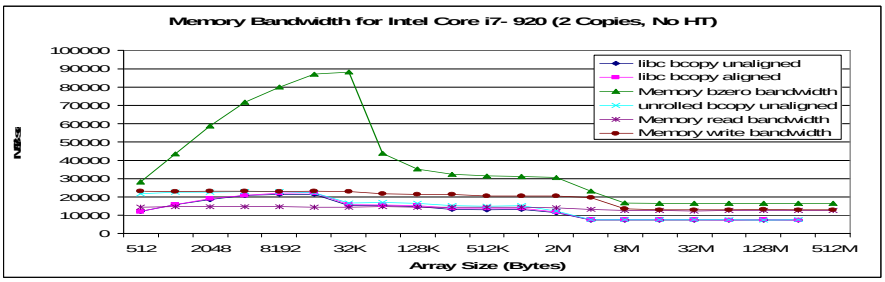


(d)

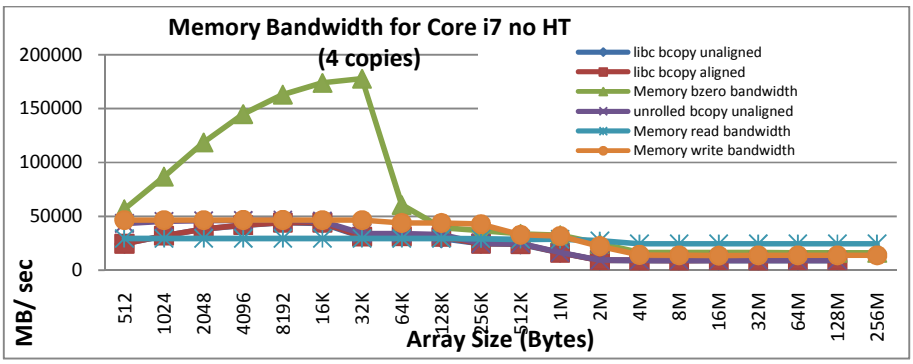
Figure 3.11 Memory Bandwidth Results for Xeon Quad Core using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies



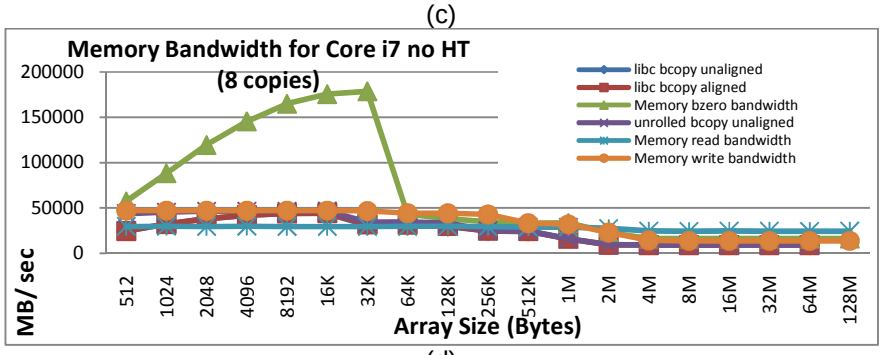
(a)



(b)

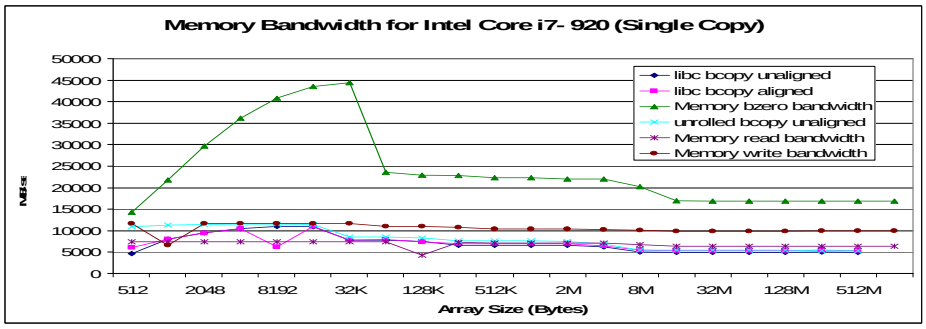


(c)

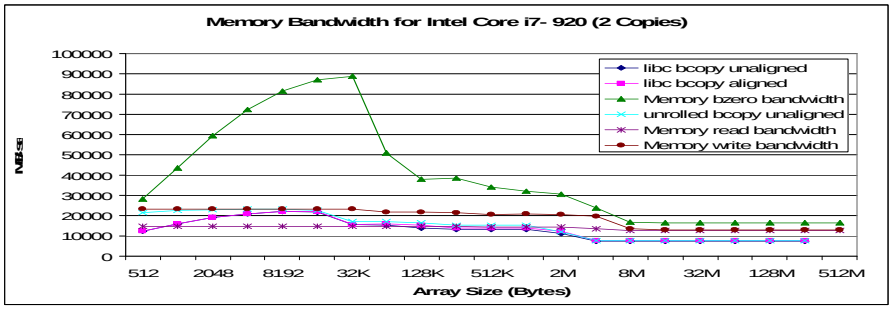


(d)

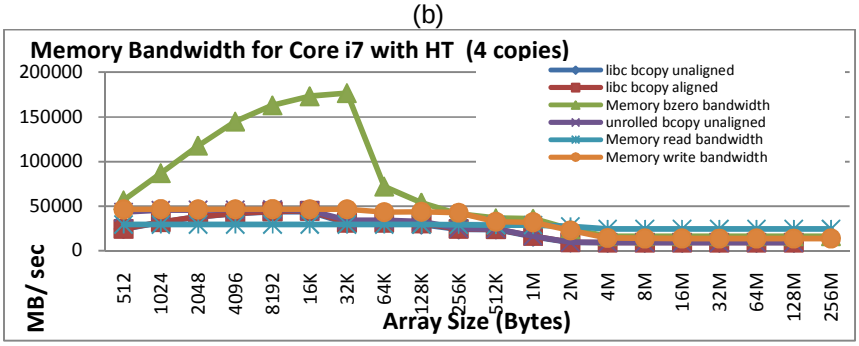
Figure 3.12 Memory Bandwidth Results for Core i7 (HT Disabled) using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies



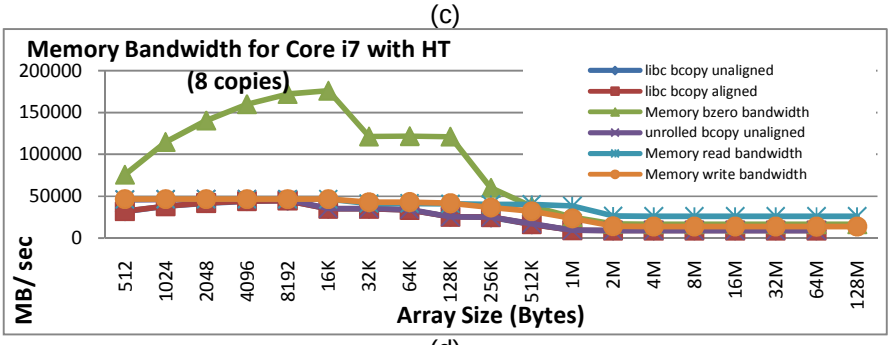
(a)



(b)



(c)



(d)

Figure 3.13 Memory Bandwidth Results for Core i7 (HT Enabled) using (a) 1 Copy, (b) 2 Copies, (c) 4 Copies, and (d) 8 Copies

3.2.4.2 STREAM Benchmark Results

Figures 3.14- 3.22 show the results obtained from running STREAM and STREAM2 benchmarks in terms of memory bandwidth. On other hand Tables 3.5- 3.8 provide a summary of best bandwidth results obtained for each of the STREAM and STREAM2 benchmark functions. The results reported in these tables are obtained from every processor tested in this work and ordered in ascending order according to the bandwidth in MB/ sec from lowest to the highest bandwidth performance for each of the benchmark functions. The tables also show the number of threads used to obtain the best bandwidth result for the given processor.

From the tables, it can be easily noticed that the tested processors can be grouped into four different groups according to the STREAM and STREAM2 benchmark performance the groups, in ascending order, are: 1. EP80579 SoC, 2. ATOM processor family, 3. Quad core processors, the Xeon and the Core 2 Quad, and finally, 4. Core i7. The performance boundaries among these groups are clear and distinctive. For instance the ATOM processor performs at least 20% to 80% better than the EP80579, while the quad core processors are at least 100% better than the ATOM processor performance; where the Core i7 performs three to four times better than the Quad core processors tested in here.

In more detail, The EP80579 SoC develops its best performance when solving the STREAM/ STREAM2 benchmark functions with the aid of three threads, except for the Triad function were the use of two threads provided best performance for solving this function.

For the ATOM processor family, the single core ATOM 230 performance was much better than the dual core ATOM 330 for the functions required either only simple memory access such as copy function from both STREAM and STREAM2 for array copy and fill function from STREAM2 for filling an array with constant or when only performing a multiplication with a constant operation like in the scale function from the STREAM benchmark. Also for these functions, it can be noticed that enabling the hyper-threading technology (HT) in either of the single or dual core ATOM worsens the processor's performance and that it is advisable to

disable the HT when executing operations which only perform similar memory access operations. Dual core ATOM outperforms the single core ATOM when executing operations that require more complicated steps like in the Add and Triad functions of the STREAM benchmark with best performance obtained when enabling HT with the aid of 16 threads. Finally, For Daxpy and sum functions of the STREAM2 benchmark both processors provides mixed performance in which single core was better than dual core ATOM in some configurations (enabling or disabling the HT technology) and vice versa.

For the quad core processors tested here, both the Core 2 Quad and the quad core Xeon L5408 processors have close performance results with some advantage for the Xeon processor over the Core 2 Quad in the Copy, Scale, and Triad functions from STREAM benchmark and Fill, Copy, and Daxpy from STREAM2 Benchmark, while the Core 2 Quad was better than the Xeon processor when running the Add and Sun functions. For all the executed functions, the Core 2 Quad best performance is obtained when running functions with three threads. While for Xeon based system the best performance of this processor is obtained when running functions with ten threads.

The Xeon processor's advantage is due to the effect of the Intel 5100 memory controller hub chipset and the use of dual Xeon processors by the development kit tested in this work. The Xeon based system performance increases linearly with increasing threads up to five threads, then it the performance decreases a little bit when increasing the number of threads for more than five but keeping the same incremental performance behavior tell reaching the peak performance with ten threads.

As for the newest Intel processor, the Core i7, the best performance of this processor is obtained by running applications with five threads with the HT technology enabled, it can also be concluded that with applications using more than five threads that the core i7 with HT disabled can provide performance equal or better than with the case of enabling the hyper threading technology.

Table 3.5 Comparison of Results from STREAM Benchmark (Copy and Scale Functions) for Tested Processors in Ascending Order

Copy			Scale		
Processor	Threads	Bandwidth (MB/sec)	Processor	Threads	Bandwidth (MB/sec)
EP80579	3	1324.79	EP80579	3	1283.15
Dual Core ATOM 330 HT Enabled	1	1625.285	Dual Core ATOM 330 HT Enabled	1	1616.975
Dual Core ATOM 330 HT disabled	1	1626.33	Dual Core ATOM 330 HT disabled	1	1620.94
Single Core ATOM 230 HT Enabled	1	1690.29	Single Core ATOM 230 HT Enabled	1	1679.1
Single Core ATOM 230 HT disabled	16	1905.538	Single Core ATOM 230 HT disabled	16	1868.338
Core 2 Quad Q6600	3	4397.015	Core 2 Quad Q6600	3	4443.72
Xeon L5408	5	4973.75	Xeon L5408	10	4918.27
Core i7-920 HT Enabled	5	19136.59	Core i7-920 HT Disabled	7	18756.43
Core i7-920 HT Disabled	6	27469.35	Core i7-920 HT Enabled	5	20548.15

Table 3.6 Comparison of Results from STREAM Benchmark (Add and Triad Functions) for Tested Processors in Ascending Order

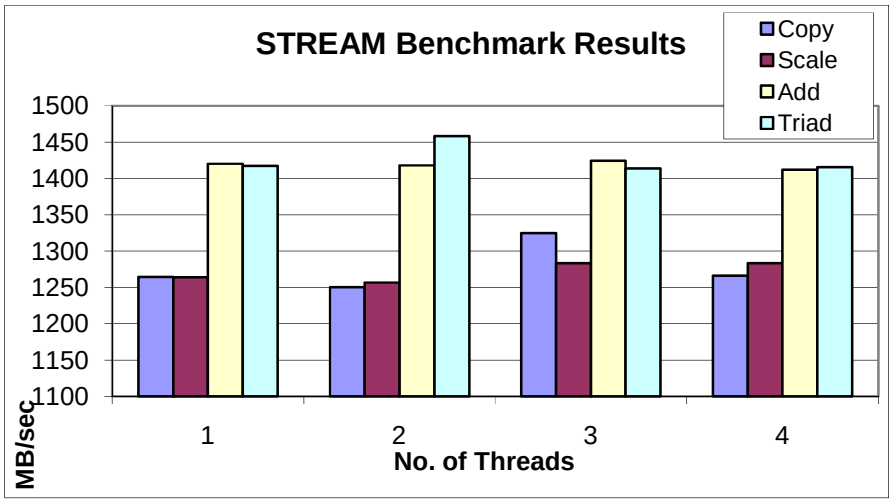
Add			Triad		
Processor	Threads	Bandwidth (MB/sec)	Processor	Threads	Bandwidth (MB/sec)
EP80579	3	1424.36	EP80579	2	1458.64
Single Core ATOM 230 HT disabled	1	1985.33	Single Core ATOM 230 HT disabled	1	1841.995
Single Core ATOM 230 HT Enabled	1	1986.445	Single Core ATOM 230 HT Enabled	1	1850.75
Dual Core ATOM 330 HT disabled	1	2077.558	Dual Core ATOM 330 HT disabled	1	1956.133
Dual Core ATOM 330 HT Enabled	16	2344.323	Dual Core ATOM 330 HT Enabled	16	1988.665
Xeon L5408	10	4917.64	Core 2 Quad Q6600	3	5064.6
Core 2 Quad Q6600	3	5074.89	Xeon L5408	10	5406.57
Core i7-920 HT Disabled	7	18635.48	Core i7-920 HT Disabled	8	20471.8
Core i7-920 HT Enabled	5	20851.99	Core i7-920 HT Enabled	5	23519.95

Table 3.7 Comparison of Results from STREAM2 Benchmark (Fill and Copy Functions) for Tested Processors in Ascending Order

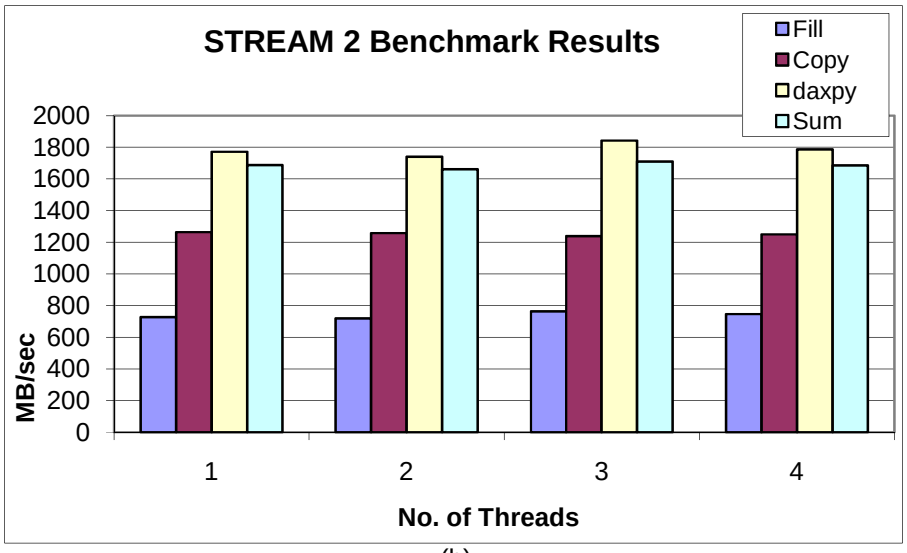
Fill			Copy		
Processor	Threads	Bandwidth (MB/sec)	Processor	Threads	Bandwidth (MB/sec)
EP80579	3	763.74	EP80579	3	1237.59
Dual Core ATOM 330 HT Enabled	1	1090.225	Dual Core ATOM 330 HT Enabled	1	1624.585
Dual Core ATOM 330 HT disabled	1	1090.595	Dual Core ATOM 330 HT disabled	1	1626.94
Single Core ATOM 230 HT Enabled	8	1127.508	Single Core ATOM 230 HT Enabled	1	1689.21
Single Core ATOM 230 HT disabled	16	1199.533	Single Core ATOM 230 HT disabled	16	1899.663
Core 2 Quad Q6600	3	3236.135	Core 2 Quad Q6600	3	4447.985
Xeon L5408	10	3566.8	Xeon L5408	10	4927.06
Core i7-920 HT Enabled	5	16826.53	Core i7-920 HT Enabled	7	18888.06
Core i7-920 HT Disabled	8	26267.24	Core i7-920 HT Disabled	10	20515.94

Table 3.8 Comparison of Results from STREAM2 Benchmark (Daxpy and Sum Functions) for Tested Processors in Ascending Order

Daxpy			Sum		
Processor	Threads	Bandwidth (MB/sec)	Processor	Threads	Bandwidth (MB/sec)
EP80579	3	1842.65	EP80579	3	1710.7
Single Core ATOM 230 HT disabled	16	2025.975	Single Core ATOM 230 HT disabled	16	2041.785
Dual Core ATOM 330 HT Enabled	2	2222.23	Dual Core ATOM 330 HT Enabled	2	2952.735
Dual Core ATOM 330 HT disabled	2	2227.98	Single Core ATOM 230 HT Enabled	8	2999.854
Single Core ATOM 230 HT Enabled	8	2463.992	Dual Core ATOM 330 HT disabled	3	3090.32
Core 2 Quad Q6600	3	6657.87	Xeon L5408	10	6102.85
Xeon L5408	10	7263.77	Core 2 Quad Q6600	3	7696.365
Core i7-920 HT Disabled	10	14302.18	Core i7-920 HT Disabled	16	18453.38
Core i7-920 HT Enabled	8	27233.35	Core i7-920 HT Enabled	5	27636.71

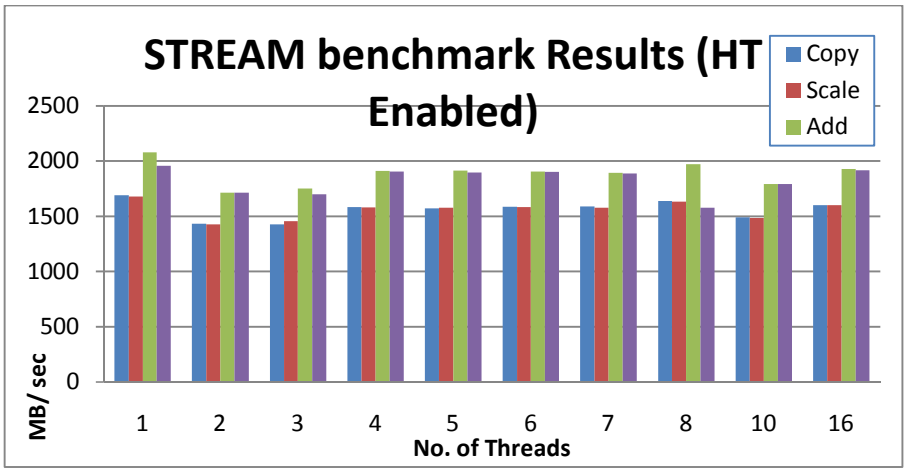


(a)

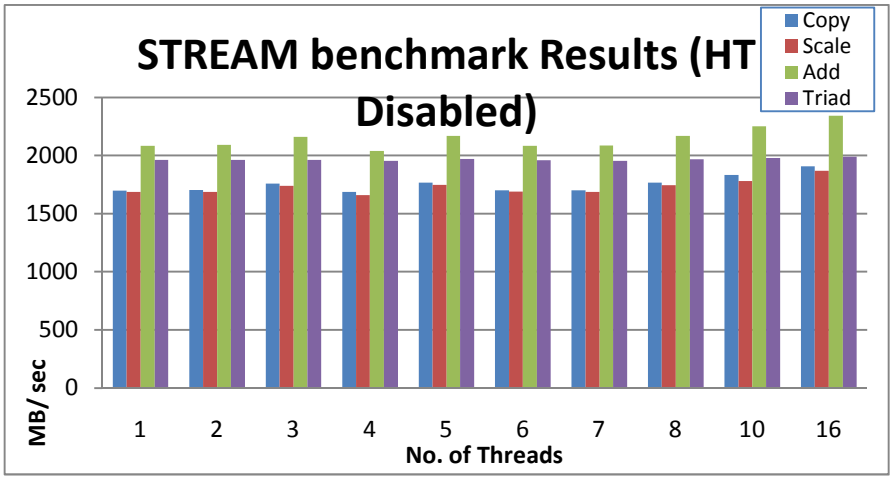


(b)

Figure 3.14 Results for EP80579 SoC (a) STREAM, and (b) STREAM2 Benchmarks

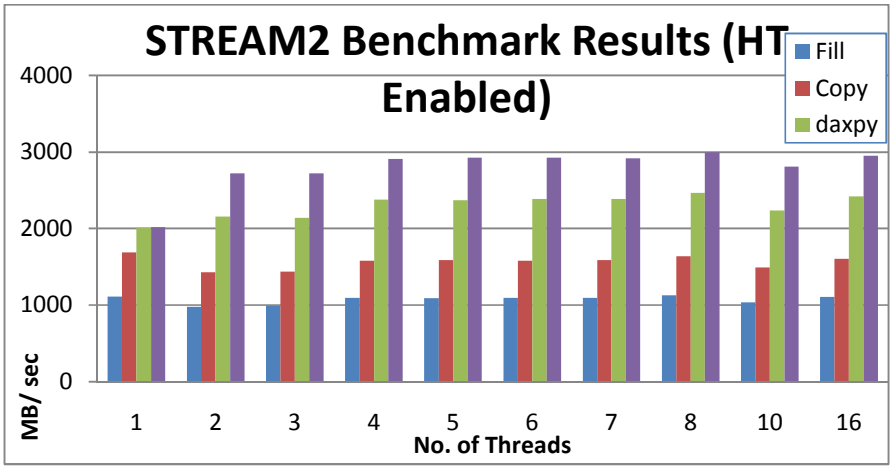


(a)

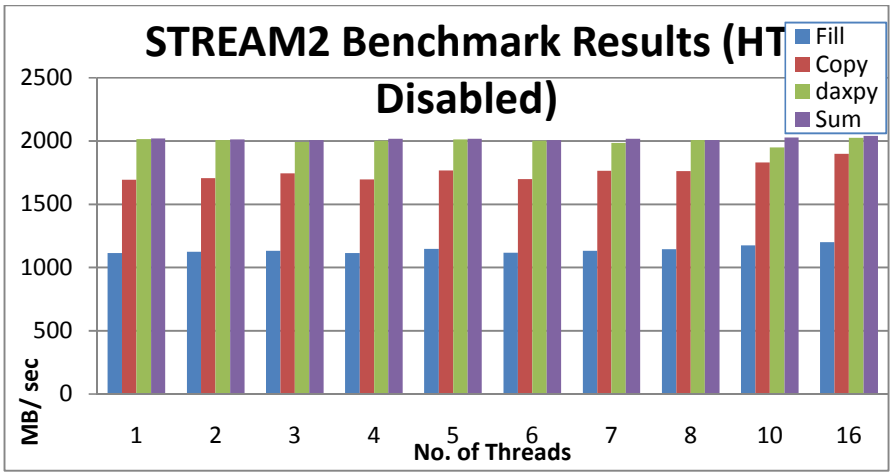


(b)

Figure 3.15 STREAM Results for Single Core ATOM-230 Processor: (a) HT Enabled, and (b) HT Disabled

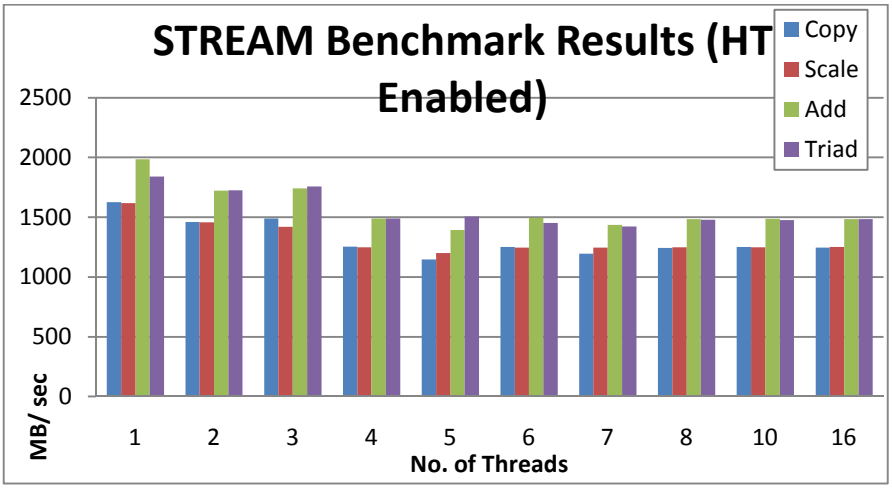


(a)

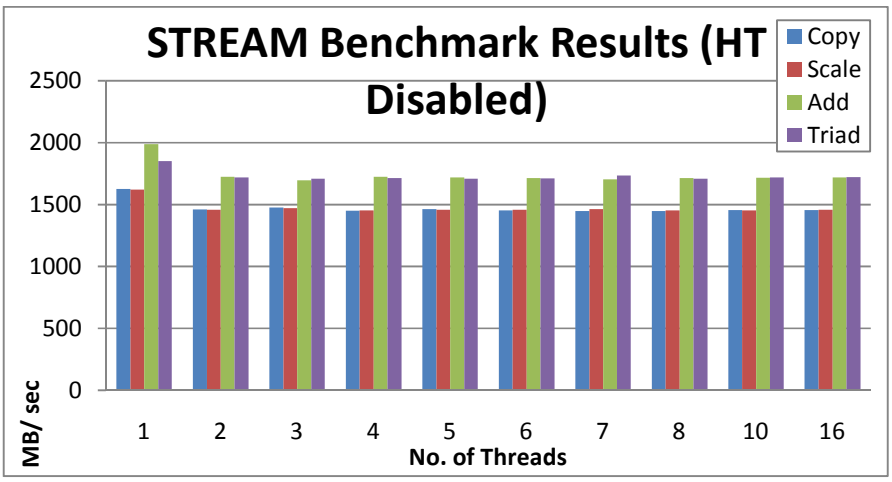


(b)

Figure 3.16 STREAM2 Results for Single Core ATOM-230 Processor: (a) HT Enabled, and (b) HT Disabled

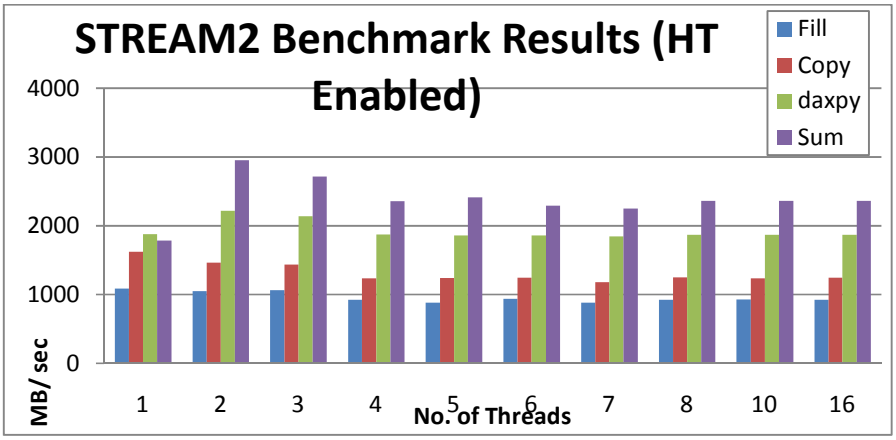


(a)

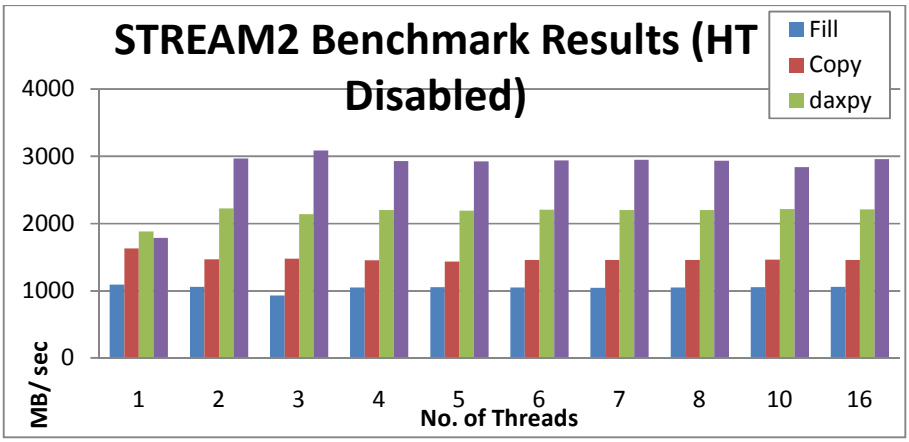


(b)

Figure 3.17 STREAM Results for Dual Core ATOM-330 Processor: (a) HT Enabled, and (b) HT Disabled

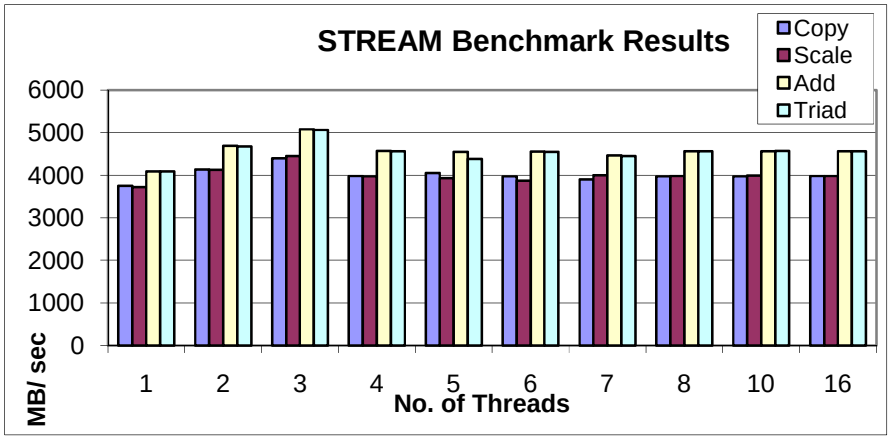


(a)

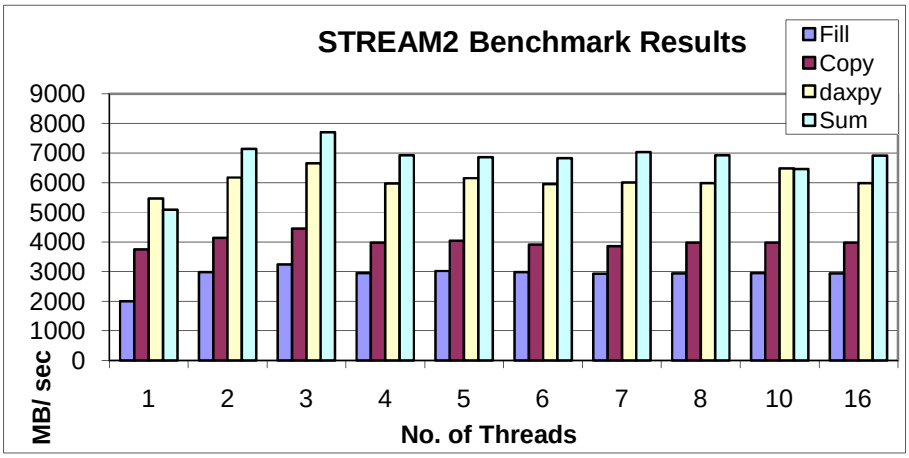


(b)

Figure 3.18 STREAM2 Results for Dual Core ATOM-330 Processor: (a) HT Enabled, and (b) HT Disabled

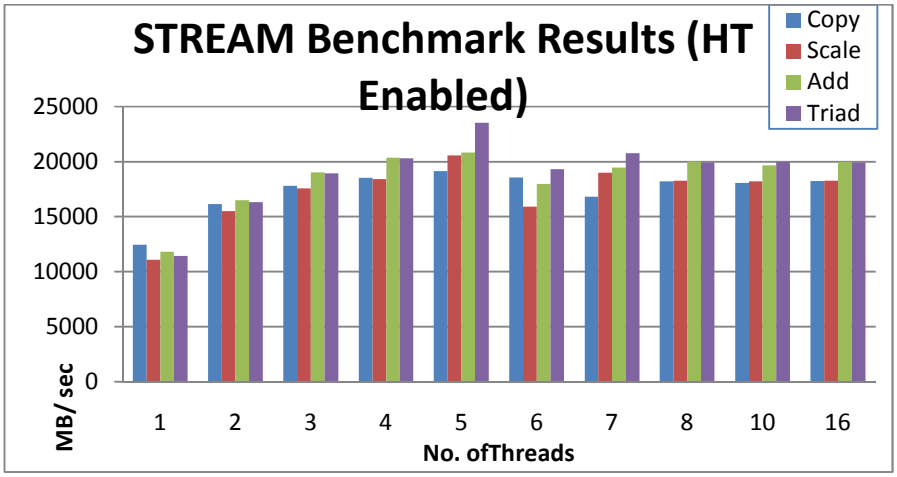


(a)

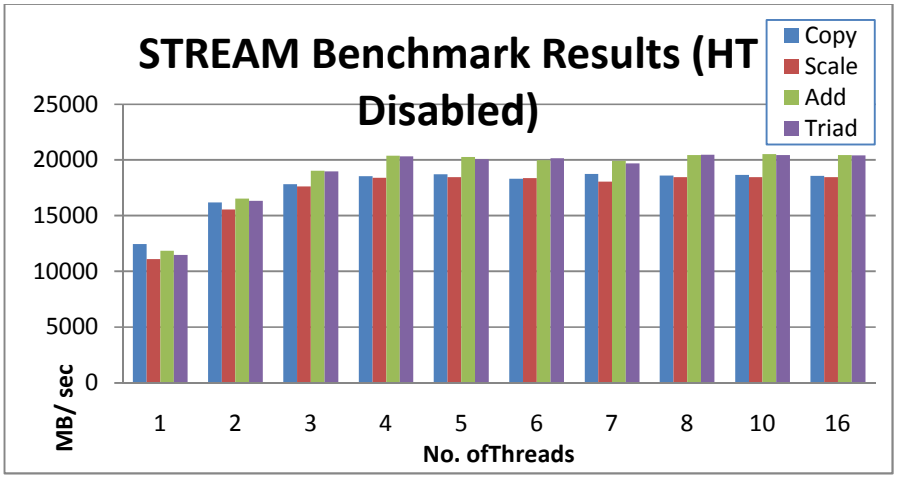


(b)

Figure 3.19 Results for Core 2 Quad Processor: (a) STREAM, and (b) STREAM2 Benchmarks

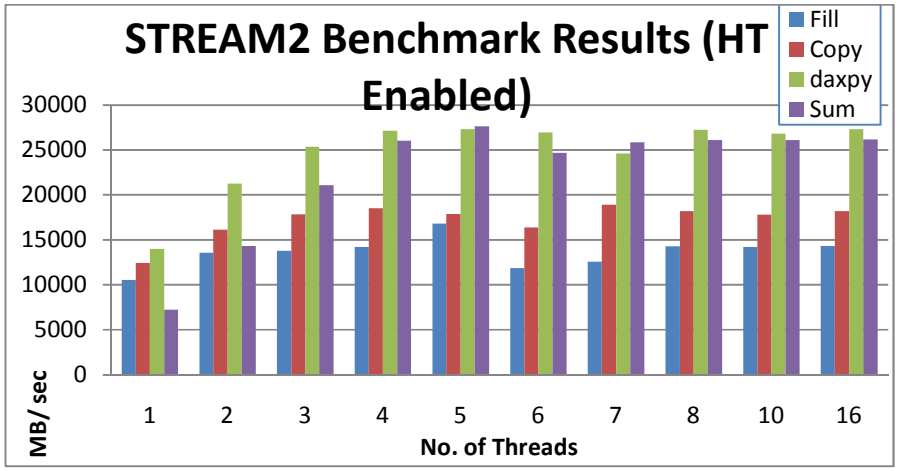


(a)

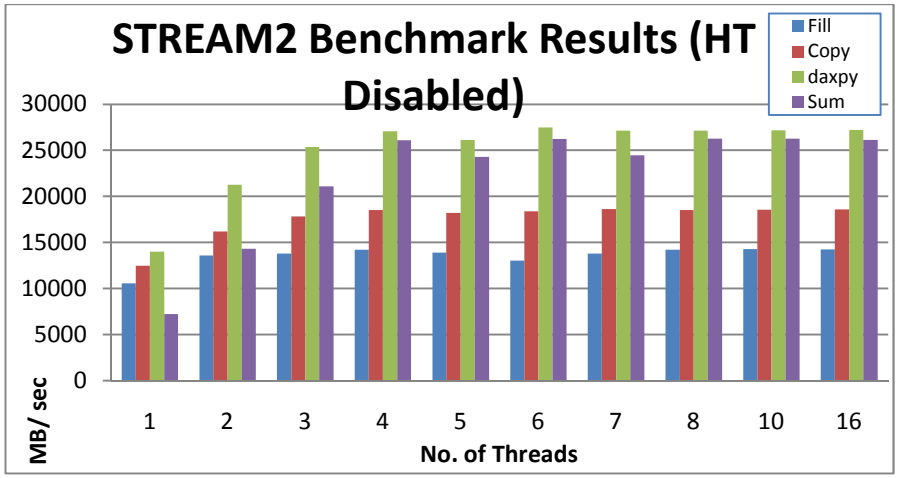


(b)

Figure 3.20 STREAM Results for Core i7-920 Processor: (a) HT Enabled, and (b) HT Disabled

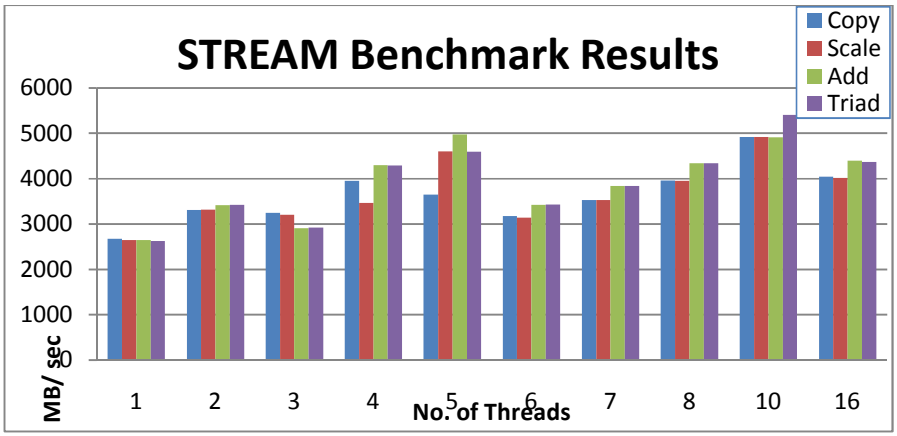


(a)

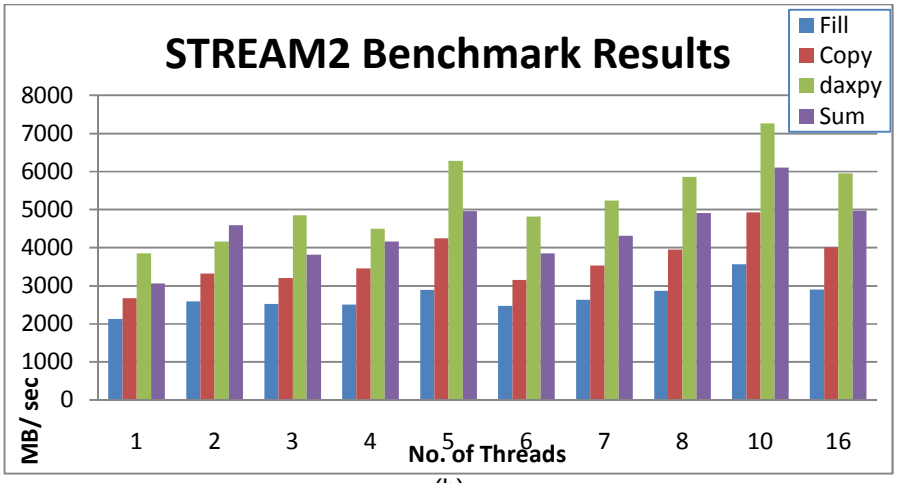


(b)

Figure 3.21 STREAM2 Results for Core i7-920 Processor: (a) HT Enabled, and (b) HT Disabled



(a)



(b)

Figure 3.22 STREAM and STREAM2 Results for Dual Xeon L5408 Processor: (a) STREAM, and (b) STREAM2 Benchmarks

3.2.4.3 NPB Benchmark Suit Results

The results were collected after running NPB version 3.3 from only eight of the eleven benchmarks in the suit are reported here. The reported benchmarks are BT, CG, EP, FT, IS, SP, UA, and LU. We used the NPB-OMP which is a sample OpenMP implementation based on the sequential implementation of the serial NPB. In this subsection we are only reporting results obtained for class B problem size. The results are split into several figures according to the

benchmark function and the processor type to be easier to read. The benchmarks were compiled using gcc 4.3 compiler with the optimization option of '-o3' and '-fopenmp' for OpenMP support. The results reported here are for the EP80579 SoC, ATOM processor (single and dual core), Core 2 Quad, and the Core i7.

Speedup was computed from the results obtained based on equation (3-1). Speedup values can clarify the gain obtained by solving any of the problems (benchmarks) with the aid of parallel processing by using threads with respect to the execution time obtained from solving the problem serially (using single thread).

$$Speedup = \frac{T_1}{T_N} \tag{3-1}$$

Where

T_1 is the execution time using single thread

T_N is the execution time using N threads

Tables 3.9 through 3.16 lists the best execution time as well as the speedup and the number of threads used for that given execution time for each of the tested processors and processor configurations. Each table reports the results for one of the benchmarks; the results are sorted in descending order according to execution time from the processor with the slowest time first up to the processor with the fastest execution time for that benchmark.

Figures 3.23 to 3.32 show the execution time required to solve the benchmarks for the tested processors.

Comparing the benchmark results shown in the figures, it can be noticed that all tested processors have the same behavior. For any given benchmark of the processor's performance improves when increasing the number of used threads until it reaches a saturation point in which the performance remains the same or drops slightly when increasing the number of threads. This is true for seven out eight of the tested benchmarks from the NPB benchmark suit.

The LU benchmark shows different and an interesting behavior, in which, for any of the evaluated processors, the performance is improved when using multi-threads as long as the thread count used is less than or equal to the available cores (logical and physical) the processor have. But when using more threads the performance worsens rapidly even comparing it to case of using single thread. For instance in the case of the core i7 processor when the HT is enabled the performance is enhanced with the increment of the threads used until reaching eight threads and after that the performance drops when trying to increase the number of threads.

This behavior can be caused by the high dependency among the threads used for problem solving since the LU benchmark is a simulated computation fluid dynamics application that uses symmetric successive over-relaxation (SSOR) method to solve a seven-block-diagonal system resulting from finite-difference of the Navier-Stokes equations in 3-D. This is accomplished by splitting it into block Lower and Upper triangular systems. There are at least two methods to implement LU in parallel: hyper-plane and pipelining [27]. Both methods of parallelization generate highly dependent parallel sections, in which certain sections cannot be processed unless the results from previous sections are ready. This kind of dependency will force some of the threads/cores to wait until another thread is done with its part. In this case with the usage of threads equal or less than the available cores, all threads and local data for the threads are loaded in the assigned core and its local cache and will be in a wait state. When using more threads than the available cores in order to execute all threads concurrently cores will perform context switching to switch from one thread to another by saving the current thread's register conditions and results. The second thread's data sets and registers are then reloaded. This is done once the execution time assigned for a thread is expired. In the case that the second thread is dependent on the results from thread one, if thread one execution is not done and its time is expired and the core switches to thread two, then this thread will be in a

wait state and waste the core's time since the required data is not ready. This will lead to a dramatic drop in the performance which was noticed in the LU results.

From Table 3.9 through Table 3.16, it is noticed that the processors can be classified into two groups. 1. The low end (low power) processors' group consisting from the ATOM processor and the EP80579 SoC, and 2. The high end processors group which consists of the Core 2 Quad and the Core i7. The high end processors group's performance is at least twice as much as the performance of the first group. The following notes can be drawn from the tables:

1. In general from all benchmarks, the core i7 has the best performance among all processors.
2. The Hyper-threading technology supported by the ATOM and core i7 processors provided mixed behavior, in which it improved the processors performance for some of the benchmarks, while the performance improved when disabling hyper-threading for some of the benchmarks like in the case of IS and SP benchmarks.
3. Comparing the dual core ATOM 330 to the single core ATOM 230 processor for both enabling and disabling hyper-threading, the performance of dual core ATOM is enhanced by at least 25% over the single core ATOM, in the case of SP benchmark with enabling hyper-threading. While the performance almost doubled from single core to dual core ATOM processor in the case of EP and IS benchmarks.

4. The EP80579 SoC performance was close to the performance delivered by the ATOM processors. Although the EP80579 has a single core processing unit, but it outperformed the single core ATOM processor even with the support of hyper-threading in BT and SP benchmarks. EP80579 also outperform the dual core ATOM when disabling the HT according to BT benchmark. For IS, LU, and UA benchmarks, EP80579 has better performance than the single core ATOM when disabling hyper-threading. This note can be due to the fact that the EP80579 has bigger L1 data cache than the ATOM processor (EP80579 has 32 KB L1 data

cache to 24 KB for the ATOM processors) and for the lack of the out-of-order execution unit from the ATOM processors (to lower the power consumption of the processor).

Table 3.9 Comparison of Best BT Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

BT			
Processor	Threads	Time (sec)	Speedup
Single Core ATOM 230 HT disabled	1	3412.41	1
Single Core ATOM 230 HT Enabled	2	2041.86	1.664786028
Dual Core ATOM 330 HT disabled	2	1821.89	1.939743892
EP80579	1	1305.16	1
Dual Core ATOM 330 HT Enabled	4	1122.32	3.15139176
Core 2 Quad Q6600	4	166.15	3.163647307
Core i7-920 HT Disabled	4	94.65	3.787427364
Core i7-920 HT Enabled	4	94.25	3.808700265

Table 3.10 Comparison of Best CG Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

CG			
Processor	Threads	Time (sec)	Speedup
Single Core ATOM 230 HT disabled	16	1515.18	2.118348975
Dual Core ATOM 330 HT disabled	2	969.66	3.034156302
EP80579	4	929.23	1.052979348
Single Core ATOM 230 HT Enabled	4	850.16	3.47460478
Dual Core ATOM 330 HT Enabled	8	583.61	5.051986772
Core 2 Quad Q6600	4	50.95	2.423552502
Core i7-920 HT Disabled	8	22.98	3.355053191
Core i7-920 HT Enabled	8	21.89	3.474189127

Table 3.11 Comparison of Best EP Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

EP			
Processor	Threads	Time (sec)	Speedup
EP80579	3	649.32	1.006037085
Single Core ATOM 230 HT disabled	1	612.69	1
Single Core ATOM 230 HT Enabled	5	347.81	1.759840143
Dual Core ATOM 330 HT disabled	2	306.66	1.997423857
Dual Core ATOM 330 HT Enabled	8	174.39	3.514134985
Core 2 Quad Q6600	4	54.15	3.980055402
Core i7-920 HT Disabled	4	39.19	3.99285532
Core i7-920 HT Enabled	16	22.42	6.982158787

Table 3.12 Comparison of Best FT Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

FT			
Processor	Threads	Time (sec)	Speedup
EP80579	4	77082.36	1.323257228
Single Core ATOM 230 HT disabled	2	550.56	1.02097864
Single Core ATOM 230 HT Enabled	7	424.21	1.309068622
Dual Core ATOM 330 HT Enabled	2	303.21	1.862240691
Dual Core ATOM 330 HT disabled	2	300.11	1.920629103
Core 2 Quad Q6600	4	34.54	2.669947887
Core i7-920 HT Disabled	4	18.65	3.618230563
Core i7-920 HT Enabled	8	18.43	3.658166034

Table 3.13 Comparison of Best IS Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

IS			
Processor	Threads	Time (sec)	Speedup
Single Core ATOM 230 HT disabled	1	22.99	1
EP80579	1	16.97	1
Single Core ATOM 230 HT Enabled	8	15.16	1.511873351
Dual Core ATOM 330 HT disabled	2	11.90	1.942857143
Dual Core ATOM 330 HT Enabled	4	7.90	2.935443038
Core 2 Quad Q6600	4	1.38	3.688405797
Core i7-920 HT Enabled	4	0.74	3.945945946
Core i7-920 HT Disabled	16	0.73	3.945205479

Table 3.14 Comparison of Best LU Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

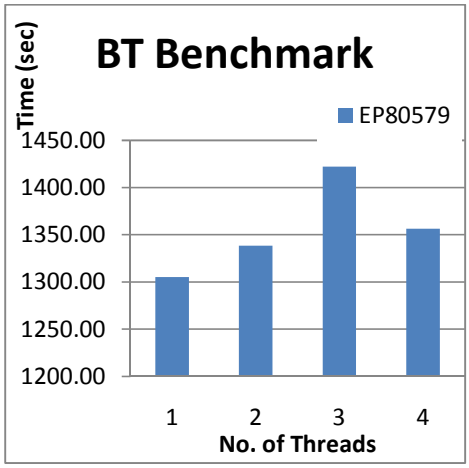
LU			
Processor	Threads	Time (sec)	Speedup
Single Core ATOM 230 HT disabled	2	2249.34	1.003912259
EP80579	1	1455.80	1
Single Core ATOM 230 HT Enabled	2	1399.4	1.595005002
Dual Core ATOM 330 HT disabled	2	1334.02	1.74968891
Dual Core ATOM 330 HT Enabled	4	898.40	2.593087711
Core 2 Quad Q6600	4	166.22	3.111478763
Core i7-920 HT Disabled	4	66.48	3.716155235
Core i7-920 HT Enabled	8	61.48	4.024886142

Table 3.15 Comparison of Best SP Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

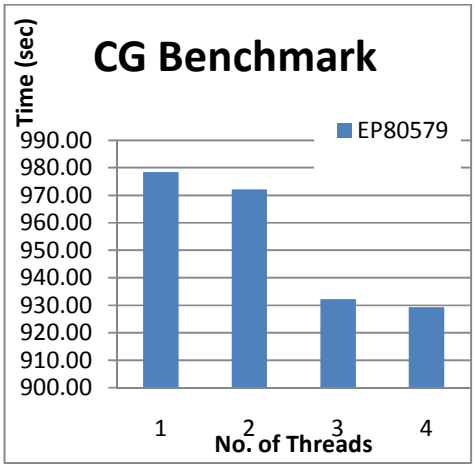
SP			
Processor	Threads	Time (sec)	Speedup
Single Core ATOM 230 HT disabled	1	2020.55	1
Single Core ATOM 230 HT Enabled	4	1362.86	1.485002128
EP80579	4	1280.23	1.006116089
Dual Core ATOM 330 HT disabled	2	1219.40	1.786435952
Dual Core ATOM 330 HT Enabled	4	1089.79	2.002670239
Core 2 Quad Q6600	4	193.19	1.844298359
Core i7-920 HT Enabled	4	72.36	3.243642897
Core i7-920 HT Disabled	4	72.32	3.237278761

Table 3.16 Comparison of Best UA Benchmark Results Obtained from Tested Processors (Sorted from Worst to Best Performance)

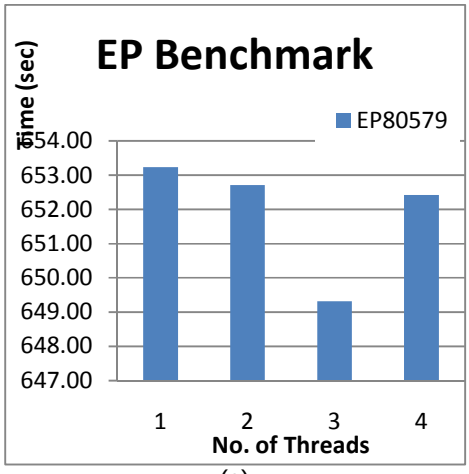
UA			
Processor	Threads	Time (sec)	Speedup
Single Core ATOM 230 HT disabled	1	1096.11	1
EP80579	1	986.35	1
Single Core ATOM 230 HT Enabled	4	796.77	1.377461501
Dual Core ATOM 330 HT disabled	2	651.64	2.079949001
Dual Core ATOM 330 HT Enabled	8	533.34	1.684595789
Core 2 Quad Q6600	8	179.03	2.331061833
Core i7-920 HT Disabled	8	63.72	3.032328939
Core i7-920 HT Enabled	16	55.14	3.576895176



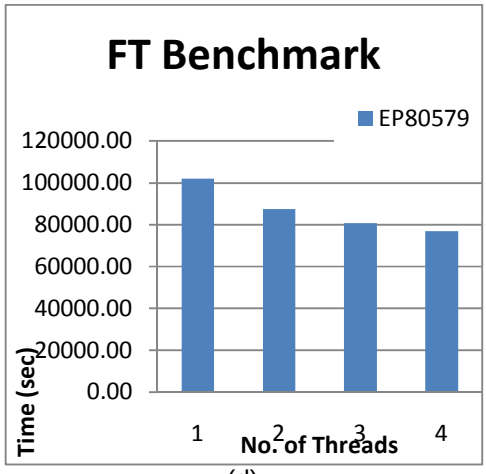
(a)



(b)



(c)



(d)

Figure 3.23 NPB Results for EP80579 SoC (a) BT, (b) CG, (c) EP, and (d) FT Benchmarks

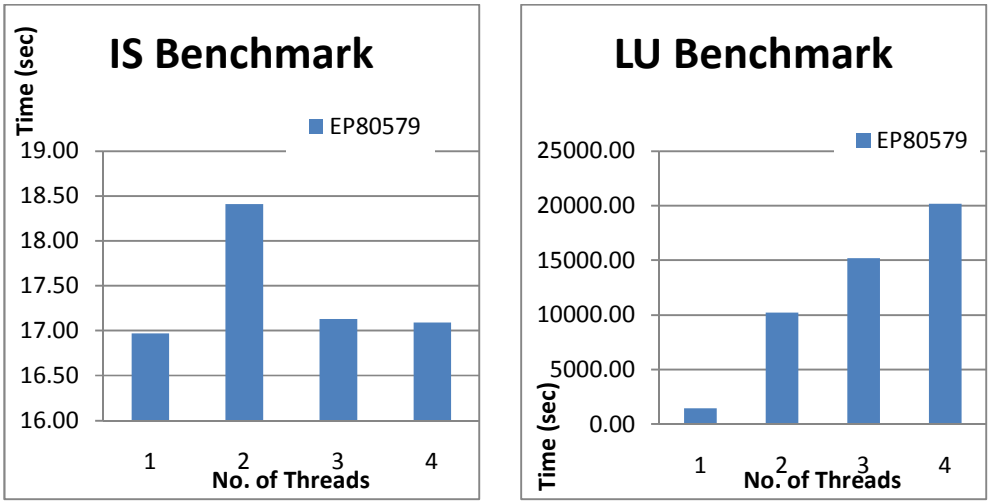
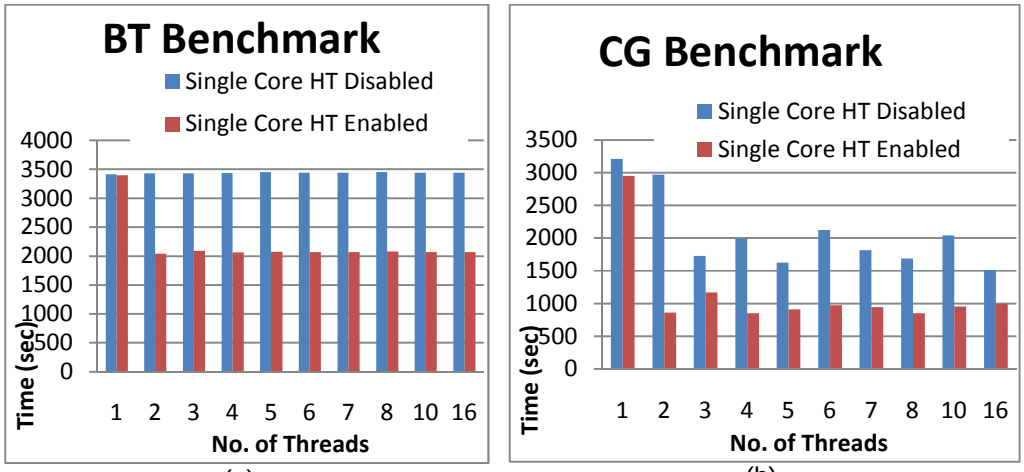
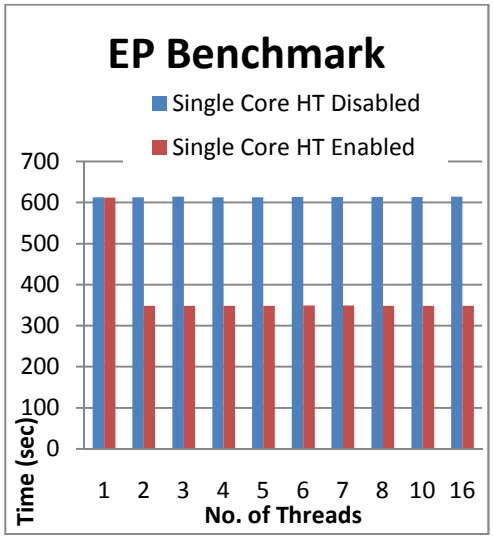


Figure 3.24 NPB Results for EP80579 SoC (a) IS, (b) LU, (c) SP, and (d) UA Benchmarks

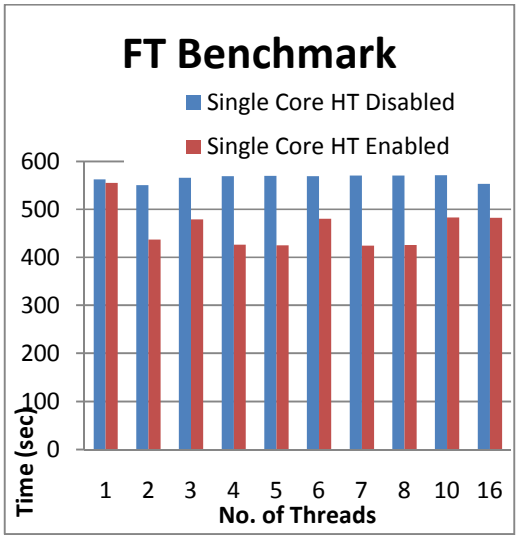


(a)

(b)



(c)



(d)

Figure 3.25 NPB Results for Single Core ATOM (a) BT, (b) CG, (c) EP, and (d) FT Benchmarks

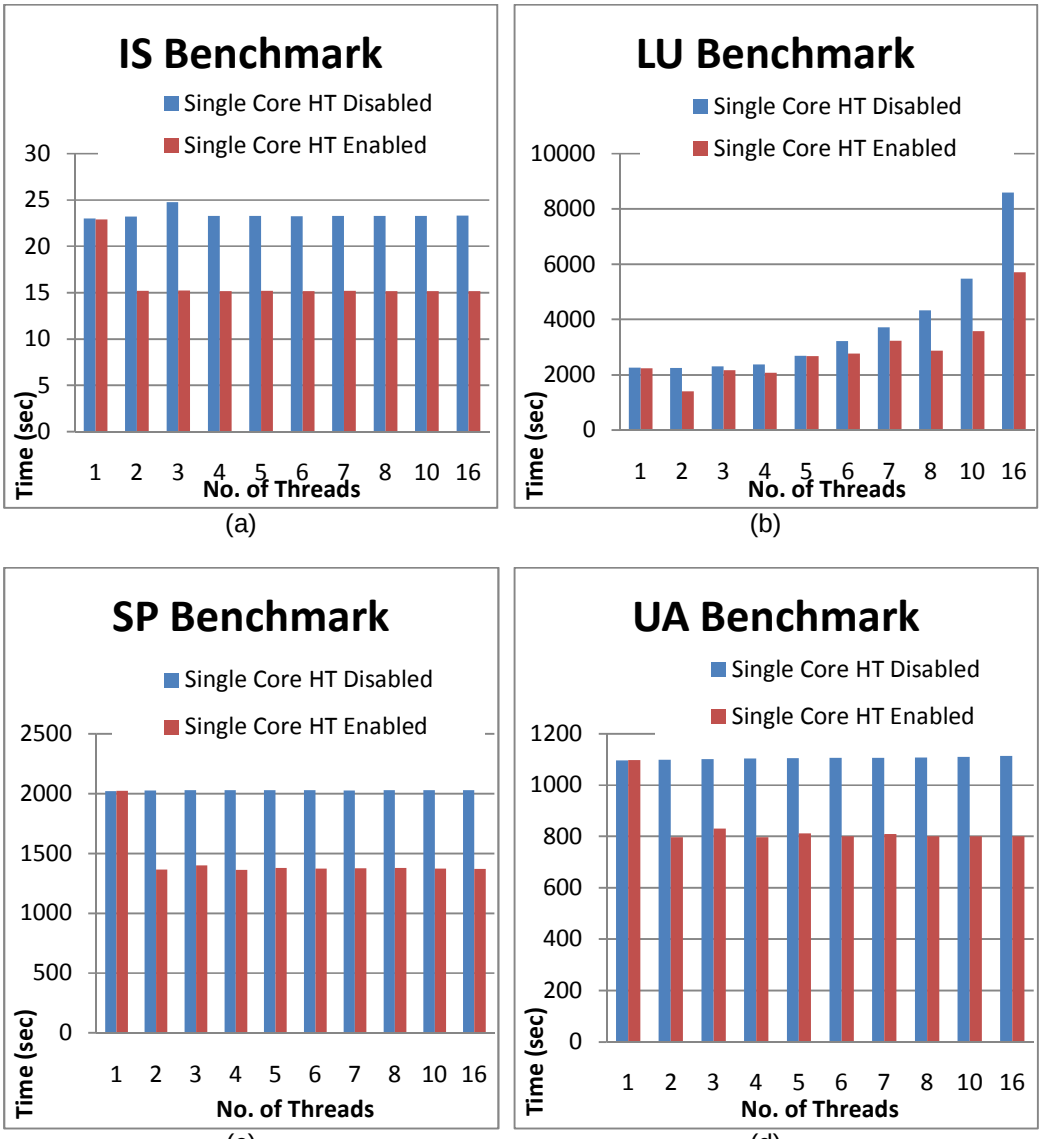


Figure 3.26 NPB Results for Single Core ATOM (a) IS, (b) LU, (c) SP, and (d) UA Benchmarks

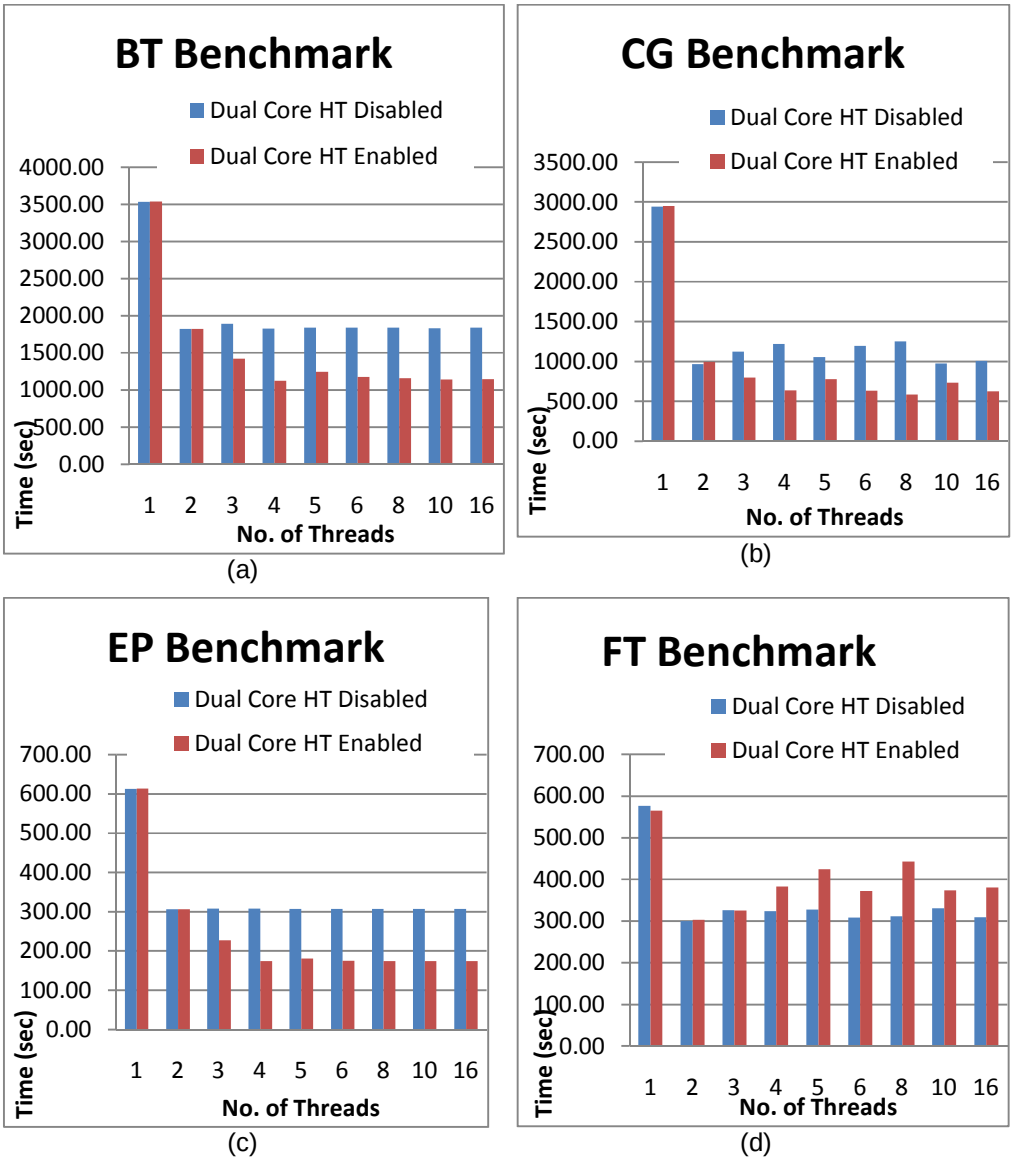
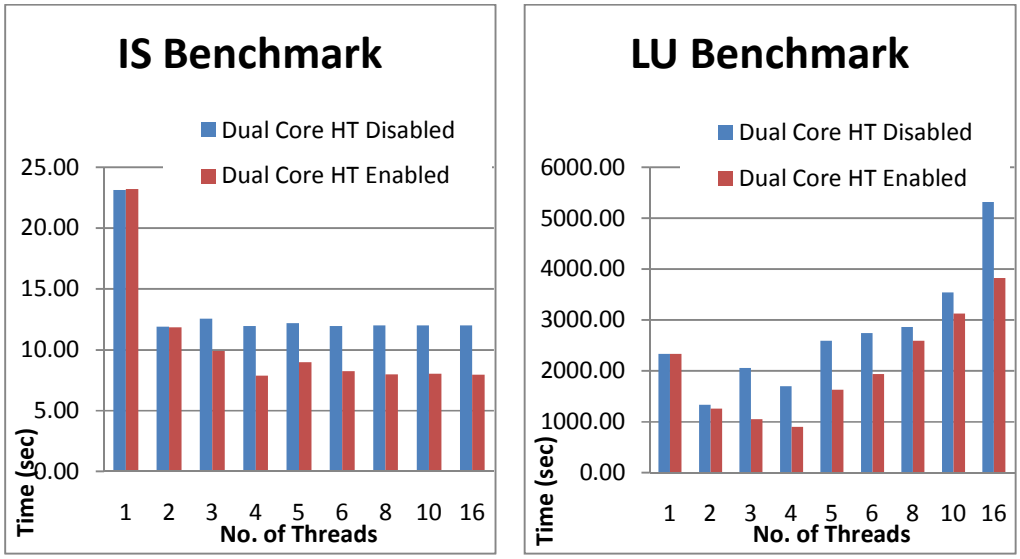
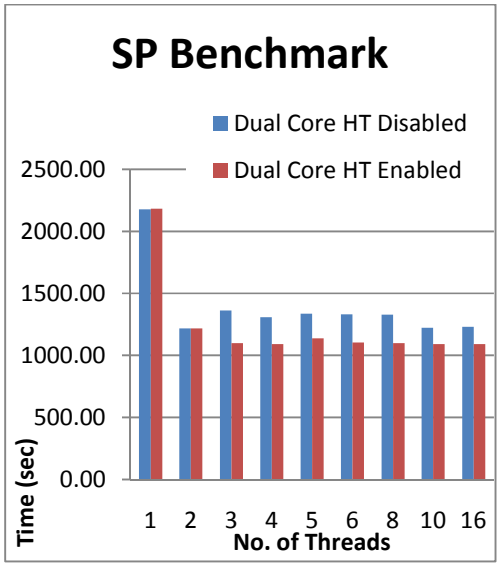


Figure 3.27 NPB Results for Dual Core ATOM (a) BT, (b) CG, (c) EP, and (d) FT Benchmarks

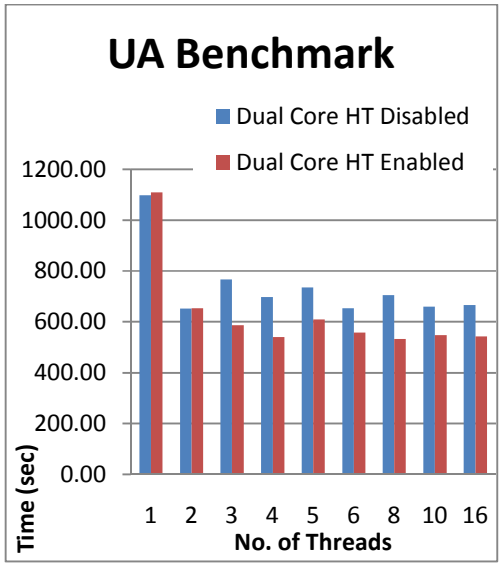


(a)

(b)



(c)



(d)

Figure 3.28 NPB Results for Dual Core ATOM (a) IS, (b) LU, (c) SP, and (d) UA Benchmarks

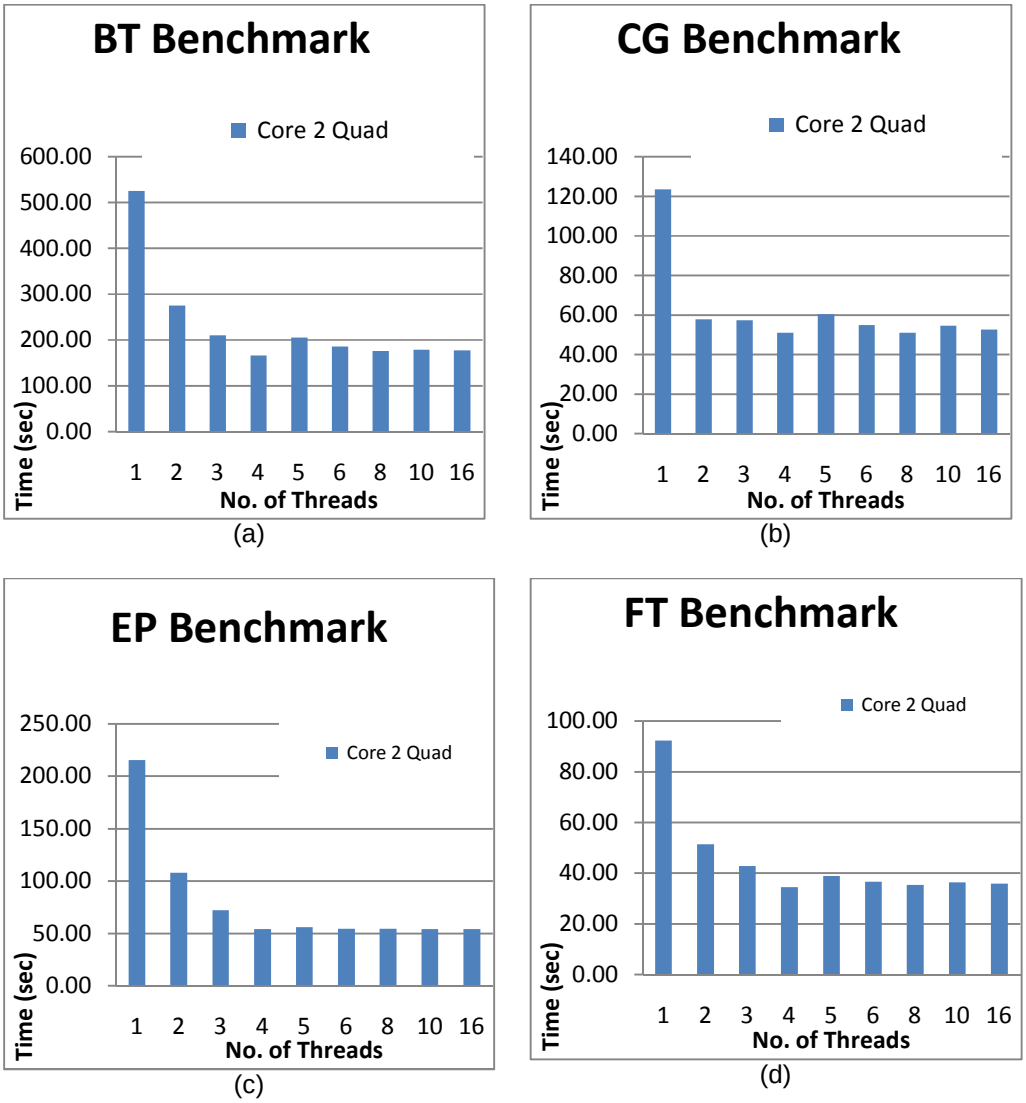
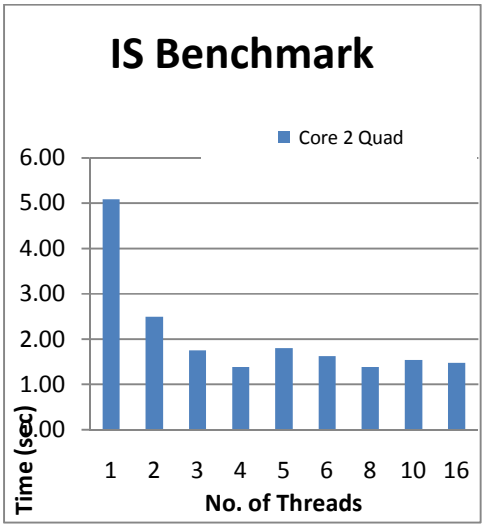
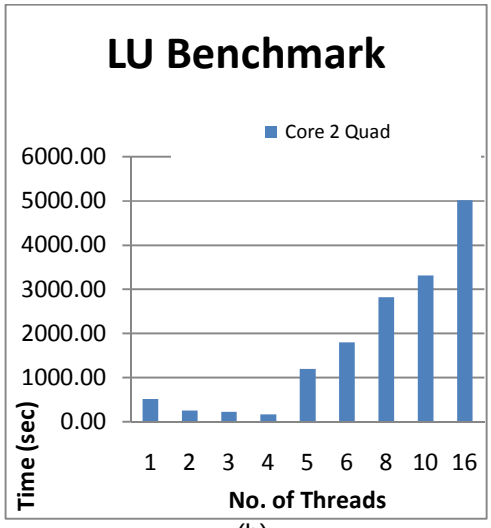


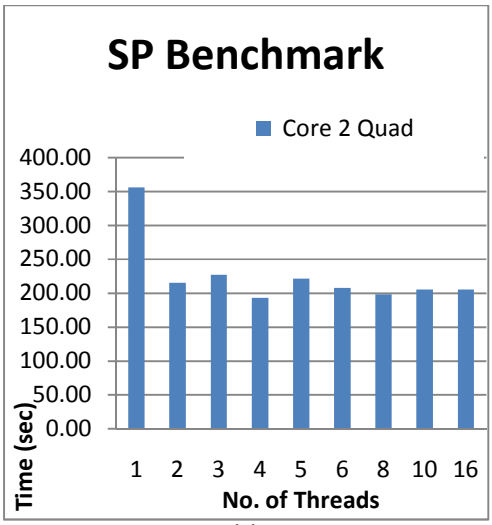
Figure 3.29 NPB Results for Core 2 Quad (a) BT, (b) CG, (c) EP, and (d) FT Benchmarks



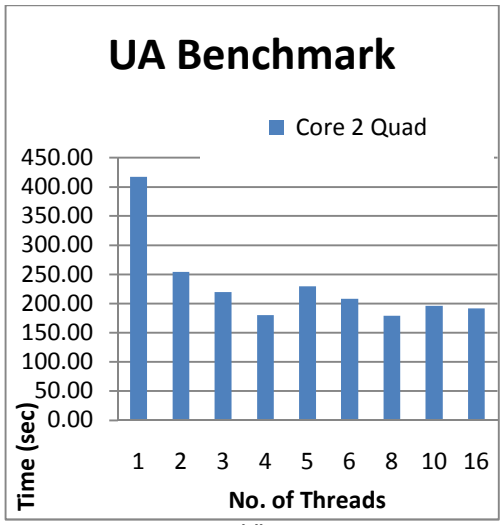
(a)



(b)

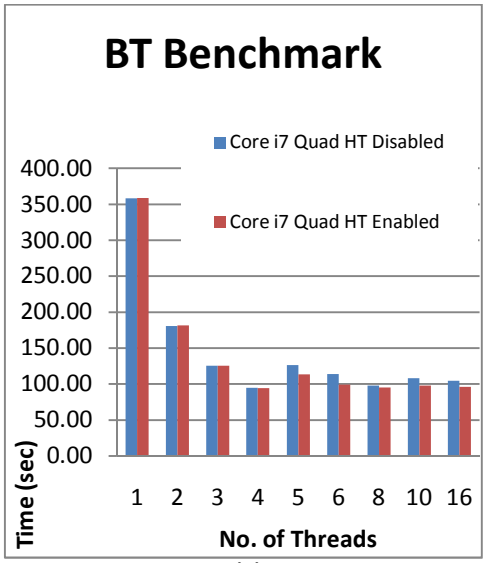


(c)

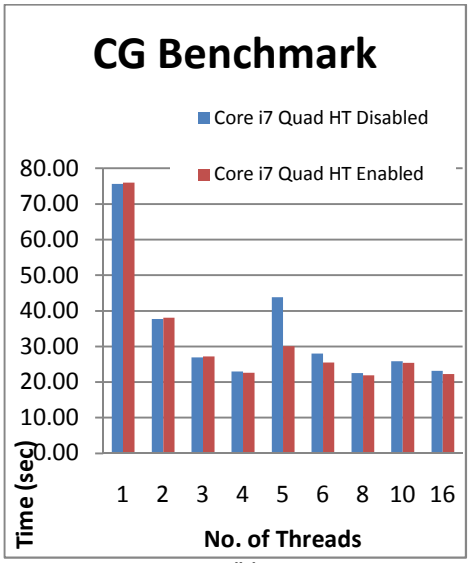


(d)

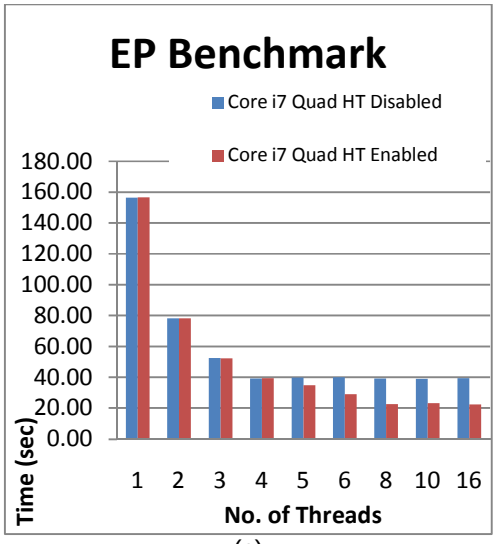
Figure 3.30 NPB Results for Core 2 Quad (a) IS, (b) LU, (c) SP, and (d) UA Benchmarks



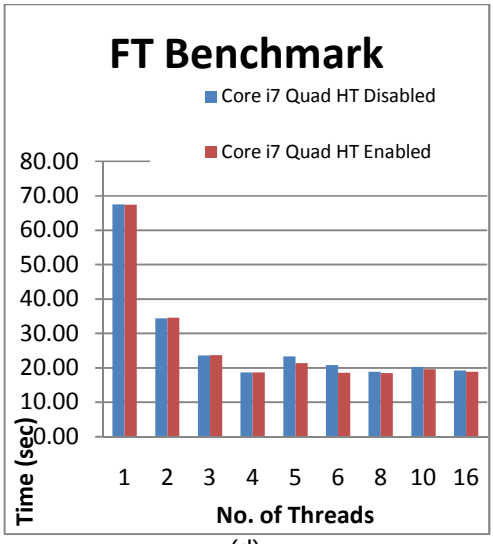
(a)



(b)

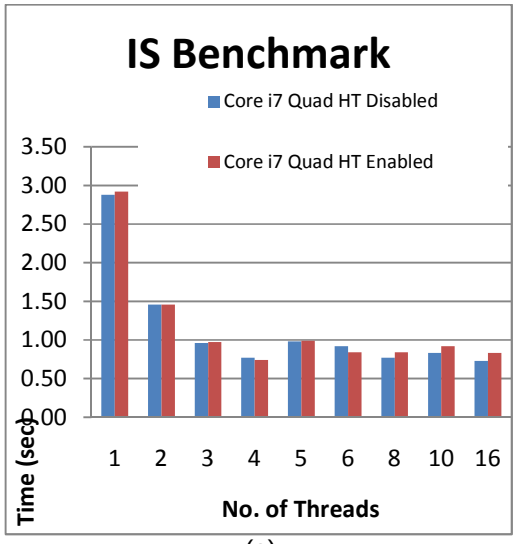


(c)

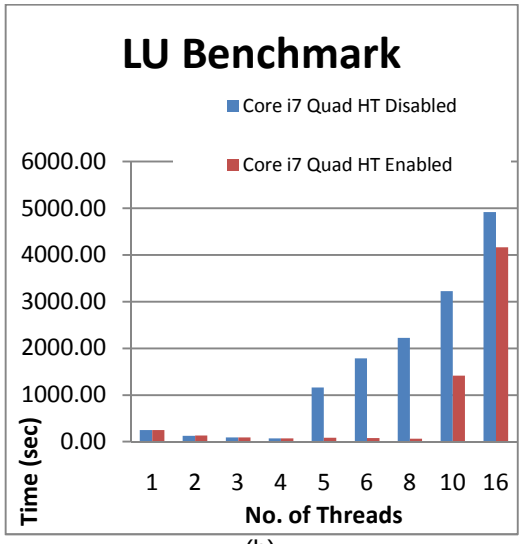


(d)

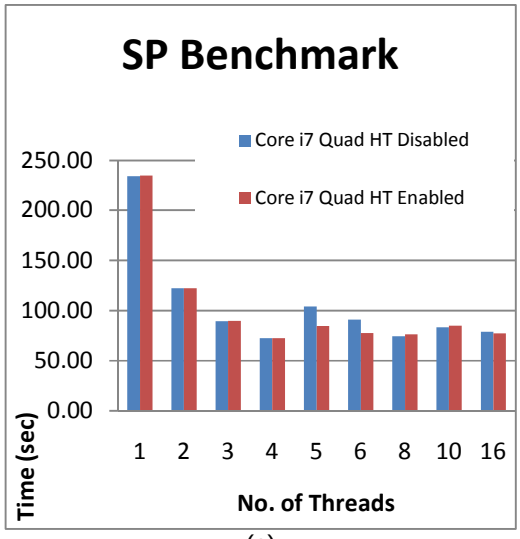
Figure 3.31 NPB Results for Core i7- 920 (a) BT, (b) CG, (c) EP, and (d) FT Benchmarks



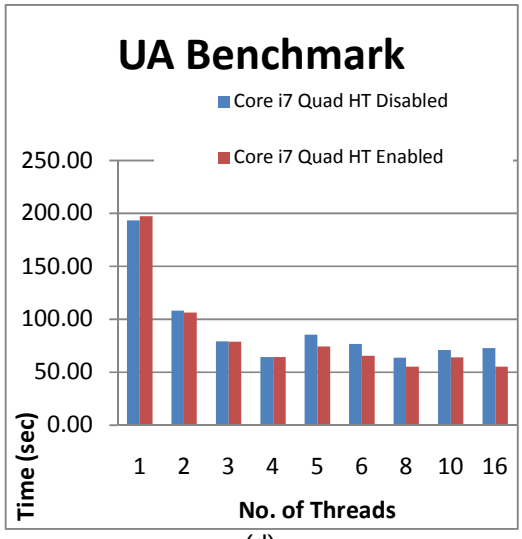
(a)



(b)



(c)



(d)

Figure 3.32 NPB Results for Core i7- 920 (a) IS, (b) LU, (c) SP, and (d) UA Benchmarks

3.2.5. Comparison of Benchmark results for different processors

Table 3.17 summaries the processors that are discussed and evaluated in this chapter and also show their characteristics. Those processors vary according to the speed, power consumption, caches levels and sizes, and also the implementation of parallelism in each one of them. were discussed then their performances were tested and analyzed using some of the common benchmark suits in terms of memory access and bandwidth, throughput, and

execution time. From the results it can be concluded that integrating the memory controller in the processors chip can increase the memory throughput as in core i7 case. Also, it shows that integrating and implementing the multi-core (CMP) technology boosts the performance of the processor more than the usage of the hyper-threading technology (SMT).

The Hyper-threading technology proves that although it can improve the performance of a given processor, but care should be taken when using (enabling) this technology since for certain applications not only it does not have any effect on increasing the performance but also it may lead to the drop of the performance.

This note is also stated by Intel. According to Intel, hyper-threading can give a performance boost of up to 30%. However, this improvement is application dependant, and there are some applications where hyper-threading actually hurts performance. [25]

Table 3.17 Comparison of the Tested Intel Processors [26]

Name	EP8057 9	Xeon L5408	Core 2 Quad Q6600	Core i7-920	Atom 230	Atom 330
# of Cores	1	4	4	4	1	2
# of Threads	1	4	4	8	2	4
Processor Base Frequency	1.2 GHz	2.13 GHz	2.4 GHz	2.66 GHz	1.6 GHz	1.6 GHz
L1 Cache	32 KB instructi on and 32 KB data	32 KB instructi on and 32 KB data per core	32 KB instructi on and 32 KB data per core	32 KB instructi on and 32 KB data per Core	32 KB instructi on and 24 KB data per core	32 KB instructi on and 24 KB data per core
L2 Cache	256 KB	12 MB	8 MB (2 X 4MB)	256 KB per Core	512 KB	1 MB (2 X 512KB)
L3 Cache	No	No	No	8 MB Intel® Smart Cache Shared	No	No
System Bus (FSB)		1066 MHz	1066 MHz	4.8 GT/s	533 MHz	533 MHz
Instruction Set	32-bit	64-bit	64-bit	64-bit	64-bit	64-bit
Processing Die Lithography	90 nm	45 nm	65 nm	45 nm	45 nm	45 nm
Max TDP	21 W	40 W	105 W	130 W	4 W	8 W
HT Technology	No	No	No	Yes	Yes	Yes
On Chip Memory Controller	Yes	No	No	Yes	No	No

CHAPTER 4

EMBEDDED SOFTWARE

Multi-core processors have become main stream in almost all types of computing devices from servers, personal computers (PCs), laptop, to embedded systems. Taking full advantage of the processing power and computational capabilities of such processors require the usage of threads in software applications for those systems. As a result, threads and threading technologies are now the main concern in the software industry since optimal software applications performance on multi-core architectures will be achieved by effectively using threads to partition software workloads.[33]

On the other hand, multi-core processors allow software developers to optimize applications by intelligently partitioning different workloads on different processor cores. An application code can be optimized to use multiple processor resources, resulting in faster application performance. [33]

In order to achieve this goal, both the Operating System (OS) installed in the computing device and the programming languages should be able to create, initialize, execute, and terminate threads as required. For instance, using a single task operating system that does not support multi-threading will degrade the performance of the overall multi-core based system because it can only deal with single processor/ core. Programming languages also have big role in implementing such software applications, especially when one or more types of threading technologies are supported in the selected language. Since using a programming language that does not support threading will lead to creating a serial (single thread) software application.

In the following sections we will discuss the Operating Systems, programming languages and libraries that enable threading implementation for embedded systems. These systems were considered for the embedded applications discussed in this research. And finally, we will discuss the threading technologies needed to design an application in order to decompose any given problem. We will then attempt to obtain the best performance with the aid of threads along with the introduction of some useful tools to check if there is any conflict among the threads and define the bottleneck points in an application.

4.1 Operating Systems (OS)

Selecting an operating system is one of the most fundamental decisions as a device manufacturer you must make. You will be looking for a platform and skilled hardware supplier that can help you meet both the technical challenges and your business requirements.

In the case of embedded systems, the operating system selected to run in such systems should have certain characteristics. These include having a small foot print since usually embedded systems do not have large storage devices. Also the operating system should be able to communicate and control any special peripheral designed specific embedded applications. Since embedded systems are mainly designed with time constraints or what is known as real-time application, the operating system must be able to meet the real-time requirements for the implemented application or the whole system will be useless.

An operating system dedicated for embedded systems is known as a real-time operating system (RTOS). RTOS is an operating system that guarantees a certain capability within a specified time constraint. Some real-time operating systems are created for a special application and others are more general purpose.

Generally, embedded systems designers' use either proprietary real-time operating system, or one of the commercially available operating systems with real-time extensions such as Linux and Windows.

In this section we will only introduce the later kinds of operating systems. The two main operating systems available for the embedded market are either Linux or Windows. In the following subsections the Linux based operating systems will be discussed followed by Windows. Finally the third subsection is for a new kind of embedded operating systems developed by Mathworks.

4.1.1. Linux Based OS for Embedded System Devices

RTOS versions of Linux are designed for devices with relatively limited resources. They are either a dedicated RTOS like LynxOS, QNX, and RTLinux, or real-time extensions added to the Linux kernel such as RTAI.

4.1.1.1 LynxOS

The LynxOS [38] RTOS is a Unix-like real-time operating system from LynuxWorks. LynxOS features full POSIX conformance and Linux compatibility. LynxOS is mostly used in real-time embedded systems.

4.1.1.2 QNX

QNX [39] is a Unix-like real-time operating system targeting the embedded market. QNX is a microkernel-based OS; it is based on running most of the operating system in the form of a number of small tasks, known as servers. This differs from the more traditional monolithic kernel, in which the operating system is a single program composed of a large number of tasks with special abilities. In the case of QNX, the use of a microkernel allows developers to turn off any functionality not required without having to change the whole operating system just the disable the server that is responsible for that task.

4.1.1.3 RTLinux

RTLinux [40] (also known as RTCore) is a hard real-time operating system microkernel that runs the entire Linux operating system as a fully preemptive process. RTLinux supports platforms like x86, PowerPC, ALPHA and others. [41]

RTLinux is based on a lightweight virtual machine where the Linux operating system serves as a guest operating system and is given a virtualized interrupt controller, timer as well direct hardware access. From the point of view of the host, the Linux kernel is a thread. Interrupts needed for deterministic processing are handled by the real-time core, while other interrupts are forwarded to Linux, running at a lower priority than real-time threads.

4.1.1.4 Real-Time Application Interface (RTAI)

RTAI [43] is a real-time extension for the Linux kernel that lets you write applications with strict timing constraints. RTAI supports different processor architectures such as x86, PowerPC, MIPS, and ARM. It provides a deterministic response to interrupts, and is POSIX compliant with native RTAI real-time tasks. RTAI consists mainly of two parts. The first one is an adaptive domain environment for operating systems **(**Adeos)-based patched to the Linux kernel which introduces a hardware abstraction layer. The second one is a broad variety of services which make real-time programmers' lives easier. [44]

4.1.2. Windows Based OS for Embedded System Devices

Windows Embedded is a family of operating systems from Microsoft designed for use in embedded systems. Windows Embedded operating systems are available to OEM system builders only, who make it available to end users pre-loaded with the hardware. The main members of Windows embedded family are Windows CE, Windows XP embedded (XPe), Windows embedded standard, and Windows embedded enterprise.

4.1.2.1 Windows CE

Windows CE supports different platforms like x86, MIPS, and ARM processor architectures. It is created and developed by Microsoft as a real-time operating system for embedded systems and was not trimmed-down or componentized from any other Windows versions. Windows CE is optimized for devices that have minimal storage. Devices are often configured without disk storage, and may be configured as a standalone system that does not allow for end-user extension. Windows CE is a preferable choice over other Windows-based

embedded operating systems in terms of the variety of processors it supports. It supports real-time applications by default and the small footprint. [37]

4.1.2.2 Windows XP Embedded

Windows XP Embedded is based on the same binaries as Windows XP Professional. It is a componentized version of Windows XP [29]. This enables embedded developers to provide the rich functionality and flexibility of the Windows XP operating system while allowing them to select only the features needed for customized, reduced-footprint embedded systems.

Windows XP Embedded does not include some of the same characteristics as Windows XP Professional. It only supports x86 architecture. So, though you can configure a build of Windows XP Embedded with the equivalent functionality of Windows XP Professional, the resulting Windows XP Embedded operating system build will not support certain features available in Windows XP Professional because these features are not required by a typical embedded device.

The major difference between Windows XP Professional and Windows XP Embedded is that Windows XP Embedded is engineered specifically to support embedded devices and their manufacturers. [29]

Windows XP embedded is more powerful and flexible than Windows CE. The biggest and foremost differentiating factor between both operating systems is that Windows CE is a completely different architecture which means that applications that are developed to be compiled into a specific format before running on a Windows CE target. While Windows CE can be as small as 300 KB, it also means that it does not have the flexibility with only 700 components to choose from when building your image. [31]

4.1.2.3 Windows Embedded Standard

Windows Embedded Standard is an updated version of Windows XP Embedded. It provides the full Win32 API and is available for x86 processors.

Windows Embedded Standard allows developers to get access to embedded specific tools that work in the familiar developer environment of Visual Studio allowing them to rapidly configure, build, and deploy devices that are more secure, reliable, and manageable.

4.1.3. xPC Target Toolbox and xPC Target Embedded Option

xPC Target [34] provides a high-performance host-target environment that enables developers to connect their Simulink and Stateflow models to physical systems and execute them in real time on low-cost PC-compatible hardware.

xPC Target allows designing a model on a host PC and then with the aid of the Real-Time Workshop toolbox to generate code. This code is then downloaded to a target PC running the xPC Target real-time kernel. The xPC target toolbox supports x86 processors. The Real-Time Workshop generates and executes stand-alone C code for developing and testing algorithms modeled in Simulink. The resulting code can be used for many real-time and non-real-time applications.

xPC Target Embedded Option [35] is an extension to xPC Target that binds the application and the xPC Target kernel into a single image. This enables the developer to automatically start an application on the target PC after the kernel boots. The created image is installed on a hard drive or flash drive on your target PC which will then load and executes the model on the target PC without a connection to a host PC.

The xPC Target Embedded Option helps in deploying the real-time embedded systems on standalone target PC hardware for production, control, signal processing, data acquisition, calibration, and test applications.

4.1.3.1 xPC Target embedded option example

In this subsection a simple real-time application will be introduced showing a possible usage of xPC target toolbox to run and control an embedded system and collect data from different analog inputs and then perform simple computations on the acquired data. A similar

system was used to support the research development effort known as a sliding profile developed and implemented as part of [53].

The application will acquire analog data from four different sensors, namely a temperature sensor, an accelerometer, a distance encoder and an opto sensor.

In order to use the xPC target toolbox and the xPC target embedded options, first the desired application is created as a Simulink model using the appropriate block sets for the target PC to be used. Most embedded boards are represented by one or more block set in the xPC target toolbox and are accessible via Simulink. Simulink runs in a host PC that is connected to the target PC over a network. In order to connect to the host PC, an xPC boot disk is used to boot the target PC as the primary operating system and enable the network communication to connect to the host PC. The network communication will be used to download the compiled Simulink model and code generated from it to the target PC. It will also be used to send commands and data back and forth between both PCs.

The target PC system used in this example is the Athena II board [45] shown in Figure 4.1 which is a PC104 based single board computer (SBC) with a data acquisition subsystem consisting of 16- single analog input channels, 4 analog output channels, and 24 programmable digital I/ O channels. Figure 4.2 shows the Simulink model created. In the upper right corner of the model there is the block set of the embedded board showing the configured analog channels that will be used in this application. Once the readings are acquired, each channel will be processed in a different manner. Then the results will be sent to the host PC (Host Scope block) and to a monitor connected to the target PC (in case of using one) for display. Figure 4.3 shows a snapshot of the output as appeared in the target PC, which not only shows a plot for signals (selected to appear with the aid of target scope block in the model) but also shows the target PC specifications such as memory usage.

Figure 4.1 Athena II Board

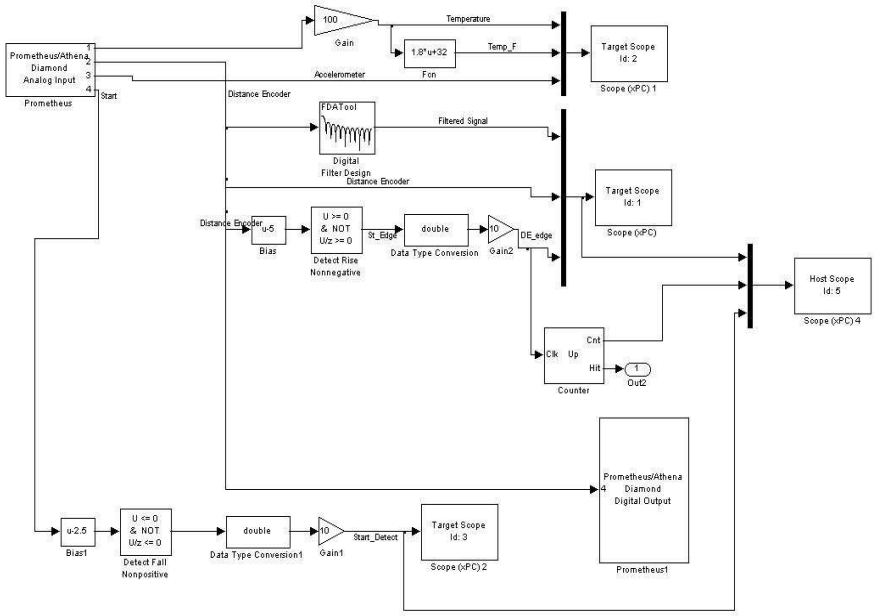


Figure 4.2 Simulink model for Embedded Application

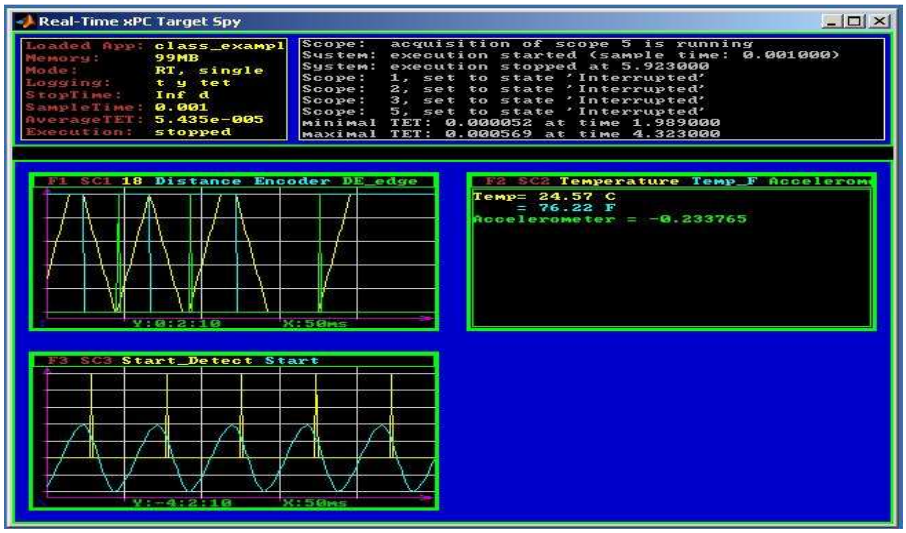


Figure 4.3 Application running on Target PC

4.2 Parallel processing capabilities of programming languages

The main practice of parallel processing involves splitting a single problem into separate parts that are solved simultaneously, allowing more work to be done faster due to the fact of using parallel execution instead of serial execution. Those parts are implemented in program units called threads, and using several threads as in the case of parallel processing is known as multi-threading.

Multi-threading is implemented within a single program, running on a single system. It requires a multi-thread capable operating system, which allows a program to split tasks between multiple threads of execution. On a machine with multiple processors, these threads can execute concurrently, potentially speeding up a given task significantly. Also multi-threading requires programming languages that provide abstractions for expressing the parallelism explicitly.

The following subsection will show different types for splitting (decomposing) a single problem into a number of parts. The second subsection discusses some of the threading techniques and the supporting libraries, while the third and last one is for introducing some of analytical tools to check the performance of multi-threading applications along with stating any

conflict among threads (if any) and defining bottlenecks in the code. Multithreading is necessary for the expanded real-time expanded road profiler analyzer applications and the portable RoLine Laser Profiler.

4.2.1. Designing for Threads

In order to split a single problem into separate parts, or create a parallel programming model from a serial programming model, developers should identify the parts (workloads) and the dependencies among them in the given program; then divide the workload to multiple threads. This division is known as decomposition.

There are three major forms of decomposition [33] and they are

1. Task (Functional) decomposition (or function parallelism) where the division is based on the type of work assigned to different thread.
2. Data decomposition (or data level parallelism) which is based on dividing the data among a number of threads that perform the same functions but processing different data block. In other words, it emphasizes loop parallelism where elements in the loop will be processed in parallel by different threads.
3. Data flow decomposition where the output of one thread is the input of another one. This decomposition type is used when there is dependency among threads which occur in what is known as a producer/ consumer problem which is the case in pipeline or wave-front programming patterns. [33], and [46]

4.2.2. Threading Techniques and support

Programming embedded multiprocessor systems is difficult. They are generally programmed at a relatively low level using C, C++, or even assembly language.

Thread implementation requires special support from the operating system and the implementation language with the support of special libraries that helps creating and manipulation threads.

Threading technologies are classified into two categories [46] library-based threads which are APIs that require linking the thread libraries to the code during compilation like Pthreads, win32 threads APIs and Intel TBB. And compiler-based threads which are a set of compiler directives like OpenMP.

In the following subsections the most wide spread threading libraries will be introduced and discussed; they are Pthreads, Win32 APIs, OpenMP, and Intel TBB.

4.2.2.1 POXIS Threads (Pthreads)

Portable Operating System Interface (POSIX) [20] is the name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces for software compatible with variants of the UNIX operating system, although the standard can apply to any operating system. POSIX defines a standard threading library API which is supported by most operating systems. This set of APIs is known as POSIX threads or Pthreads which is a portable threading library designed with the intent of providing a consistent programming interface across multiple operating system platforms. [33]

Pthreads defines a set of C programming language types, functions and constants. It is implemented with a header file (pthread.h) and a thread library.

Figure 4.5 lists a C code example for creating and running threads using Pthreads. The code first includes the supporting header file (pthread.h) then defines the maximum number of threads to be used (4 in this case) and use these numbers as thread ID that is passed to the create thread function (pthread_create). Thread creation is done here in a loop format which means that four threads will be created that runs the same function or task (Function1), but different data set will be processed by each thread depending on thread ID (defined by pthread_t type variable) and other factor that is determined in the function. This way of threading follows data decomposition technique. In case of implementing task decompistion, several (pthread_create) calls will used in the code each of them is responsible of executing

different function. Figure 4.6 lists an example of how to create threads to implement function decomposition. The pthread_create function is defined shown in Figure 4.4.

```
int pthread_create(pthread_t *restrict thread,//Thread ID
  const pthread_attr_t *restrict attr,// Thread attribute
                                      // (usually NULL)
  void *(*start_routine)(void*),      // Function name
  void *restrict arg                  // Parameter
                                     );
```

Figure 4.4 pthread_create function definition

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     4
...
void *Function1(void *threadid);
int main (int argc, char *argv[])
{
  pthread_t threads[NUM_THREADS];
  int rc;
  long t;
  for(t=0; t<NUM_THREADS; t++){
  printf("In main: creating thread %ld\n", t);
  rc = pthread_create(&threads[t], NULL, Function1, (void*)t);
     }
   pthread_exit(NULL);
}
```

Figure 4.5 Data Decomposition Example using Pthreads

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     4
...
void *Function1(void *threadid);
void *Function2(void *threadid);
void *Function3(void *threadid);
void *Function4(void *threadid);
int main (int argc, char *argv[])
{
 pthread_t t1, t2, t3, t4;
 int rc1, rc2, rc3,rc4;
 long t[4] ={1,2,3,4};
 rc1= pthread_create(&t1, NULL,Function1,(void*)t[1]);
 rc2= pthread_create(&t2, NULL,Function2,(void*)t[2]);
 rc3= pthread_create(&t3, NULL,Function3,(void*)t[3]);
 rc4= pthread_create(&t4, NULL,Function4,(void*)t[4]);

   pthread_exit(NULL);
}
```

Figure 4.6 Task Decomposition Example using Pthreads

4.2.2.2 Microsoft Win32 Thread API

Win32 thread API is a thread library for C and C++ programming languages. Threads are created by using the (CreateThread()) function that returns a handle of the thread. While the syntax is different between Pthreads and Win32 thread APIs, most of the functionality in one model will be found in the other. [46]

Figures 4.7 and 4.8 are example of implementing data and function decomposition respectively. Code resembles the way Pthreads used to create and manipulate threads with but with the use of different functions.

```
#include <windows.h>
#include <stdio.h>
const int gNumThreads = 4;
DWORD WINAPI threadFunction(LPVOID pArg);
...
void main()
{
   HANDLE threadHandles[gNumThreads];
   int tNum[gNumThreads];
   for ( int i=0; i<gNumThreads; ++i )
   {
    tNum[i] = i;
    threadHandles[i] =
    CreateThread( NULL,           // Security attributes
                  0,              // Stack size
                        threadFunction,  // Thread function
                        (LPVOID)&tNum[i],// Data for threadfunc
                        0,              // Thread start mode
                        NULL);          // Returned thread ID
          }
WaitForMultipleObjects(gNumThreads, threadHandles, TRUE, INFINITE); //
Wait Until all running threads terminate
}
```

Figure 4.7 Data Decomposition Example using Win32 Thread API

```
#include <windows.h>
#include <stdio.h>
const int gNumThreads = 4;
DWORD WINAPI t1(LPVOID pArg);
DWORD WINAPI t2(LPVOID pArg);
void main()
{
  HANDLE tH1, tH2;
  int tNum[gNumThreads];
  for ( int i=0; i<gNumThreads; ++i )
  tNum[i] = i;
  tH1 = CreateThread(NULL,0,t1,(LPVOID)&tNum[i],0,NULL);
  tH2 = CreateThread(NULL,0,t2,(LPVOID)&tNum[i],0,NULL);

WaitForMultipleObjects(gNumThreads, threadHandles, TRUE, INFINITE);
}
```

Figure 4.8 Task Decomposition Example using Win32 Thread API

4.2.2.3 OpenMP

OpenMP is an application programming interface (API) for parallel programming which consists of compiler directives and library of support functions that is configured to work with FORTRAN, C, or C++ programming languages. OpenMP supports both data and functional

Parallelism on a shared memory system. OpenMP provides support for concurrency, synchronization, and data handling while hiding the need for explicit thread management.

The OpenMP uses the pragma-based approach for writing parallel programs. The pragmas are inserted in a program to assist and guide the compiler to parallelize a given program.

Threads are first created with OpenMP when thread number is initialized to a certain number using (omp_set_num_threads(N)) function call or using (#pragma omp parallel num_threads(N)) compiler directive, where N is the number of threads.

Figure 4.9 and Figure 4.10 lists two examples for implementing function and data decomposition. In order to implement function decomposition, sections are used the (#pragma omp section) to create a number of sections where each section will be assigned to different thread assuming that there are enough threads. Each section will be executed by only one thread and threads cannot be created or reassigned inside the section. Data decomposition which is considered the main advantage of using OpenMP since it can handle this kind of decomposition in simple and fixable way; it uses the (#pragma omp parallel for) directive that will decompose for loop into number of parts equivalent to the available threads. Figure 4.10 illustrates an example of implementing data decomposition, in this example the for loop will be executed by a number of threads determined by the available processing units (cores) where each thread will handle a range of the loop determined automatically by the compiler according to the OpenMP directives. The example is for code used for positive and negative edge detection.

```
#include "stdio.h"
#include "math.h"
#include <time.h>
#include <omp.h>
...
void Function1();
void Function2();
void FunctionN();
int main()
{
// Set number of threads equivelent to number of cores/ hardware threads
omp_set_num_threads(omp_get_num_procs());
// Set fixed number of threads
//omp_set_num_threads(4);
#pragma omp parallel sections
{
 #pragma omp section
          Function1();
 #pragma omp section
     Function2();
...
 #pragma omp section
     FunctionN();
          }
 printf("Program End...\n");
}
```

Figure 4.9 Sample Code for Task (Functional) Decomposition using OpenMP

```
#include "stdio.h"
#include "math.h"
#include <time.h>
#include <omp.h>
...
int main()
{
// Set number of threads equivelent to number of cores/ hardware threads
omp_set_num_threads(omp_get_num_procs());
#pragma omp parallel for
          for (i = 0; i < d; i++)
           {
             if((distance[i] - distance[i+1]) > 5)
                    {
                              falling[j] =i;
                              j++;
                    }
             else if((distance[i] - distance[i+1]) < -5)
                    {
                              rising[k] = i;
                              k++;
                    }
           }
}
```

Figure 4.10 Data Decomposition Example using OpenMP

4.2.2.4 Intel based threading: Threading Building Blocks (TBB)

Intel Threading Building Blocks (TBB) [33], [46], and [48] is a C++ template library developed by Intel to write software programs that take advantage of multi-core processors. This library is for loop-level parallelism that concentrates on defining tasks rather than explicit threads. [49]

The Intel TBB library consists of data structures and algorithms that allow a programmer to avoid some complications arising from the use of native threading packages in which individual threads of execution are created, synchronized, and terminated manually like in the Pthreads or win32 thread APIs. This means that Intel TBB resembles OpenMP in terms of parallel programming support and hiding threading details from the programmer. Intel TBB and OpenMP are designed for threading for performance and scalability, providing constructs that emphasize scalable data parallel decomposition. [52]

The library abstracts access to the multiple processors by allowing the operations to be treated as "tasks," which are allocated to individual cores dynamically by the library's run-time engine, and by automating efficient use of the cache. Intel TBB can work with most common used operating systems from Windows based operating systems, Linux, and Mac OS.

4.3 Analysis Software packages

Intel introduced several software packages that help in analyzing applications, especially the ones that is implemented using any of the threading technologies these packages are VTune Performance Analyzer, Intel Thread Checker, and Intel Thread Profiler. Those packages analyze any application by running the application and collecting different performance and statistical information, concentrating on threads, accessing shared memory blocks, identifying race conditions among threads, and specifying bottlenecks in the application. Those analysis tools support only Intel based processors. In the following subsections and those tools will be introduced and discussed briefly. Check [50] and [51] for more information about these tools.

4.3.1. VTune Performance Analyzer

The Intel VTune Performance Analyzer offers several features that help in performance tuning, among these features are:

- Sampling: Calculates the performance of an application over a period and for various processor events.
- Call Graph: Provides a graphical view of the flow of an application and helps you identify critical functions and timing details.
- Counter Monitor: Provides system-level performance information such as resource utilization during the execution of an application. This functionality works only on Windows.
- Hotspots View: Helps identify the area of code that consumes the maximum CPU time.

- Tuning Assistant: Provides tuning advice by analyzing the performance data. The tuning advice provides a guideline for the programmer to improve the performance of an application. This functionality works only on Windows.

4.3.2. Intel Thread Checker

Is another tool used in performance analysis and work as a debugging tool used on threaded applications. Intel thread checker can detect threading bugs in Windows threads, POSIX threads, and OpenMP threaded applications. It also detects potential threading related bugs even if they do not occur.

The Intel thread checker tool is a plug-in to Intel VTune Performance Analyzer with the same look, feel, and interface as the VTune Analyzer environment. It can reduce the turnaround time for bug detection and isolation.

4.3.3. Intel Thread Profiler

Intel Thread Profiler is a threading tool that you can use to analyze the performance of applications threaded with Windows threads API, OpenMP, and POSIX threads. In Microsoft Windows, Intel Thread Profiler is a plug-in for the Intel VTune Performance Analyzer. Intel Thread Profiler can be used to perform the several functions such as Group performance data by categories such as thread and source location. Sort data according to various criteria such as type of activity time, concurrency level, or object type. And finally, identify source locations in your code that cause performance problems.

4.4 Example of using Intel Vtune for application analysis

An application is created to acquire data from 2 analog sources through the Analog to digital converter unit (ADC) that is part of the data translation module DT9816.

The data acquired is then filtered using two different filters. It uses an FIR filter with 301 coefficients and an average filter with 500 coefficients. Threads are implemented in the application. The main thread is responsible for acquiring the data using sample rate of 1000 Hz while another two separate threads are used to filter the data. Standard deviation along with the

maximum input value from both input channels are displayed every 3 seconds. When the user exits the program all the acquired data points as well as the filtered points are written to an output file.

Figures 4.11 – 4.16 show the analysis results. Figure 4.11 and Figure 4.12 illustrate call graphs representing the application under analysis. Call graph is used to collect and display information about the program flow of the application.

- Hotspots are shown in Figure 4.13. A hotspot is used to check application components activities and helps in focusing attention when looking for bottlenecks. If the bottleneck is micro-architectural, then finding a hotspot is really important. Hotspots are located after performing statistical sampling of the running applications and their components. The sampling collector periodically interrupts the processor on either time-based or event-based exceptions. It collects information regarding
    - Execution address in memory (CS:IP)
    - Operating system process and thread ID
        - Executable module loaded at that address.

Figures 4.14- 4.16 display sampling summary of functions' activities, as well as threads activities.

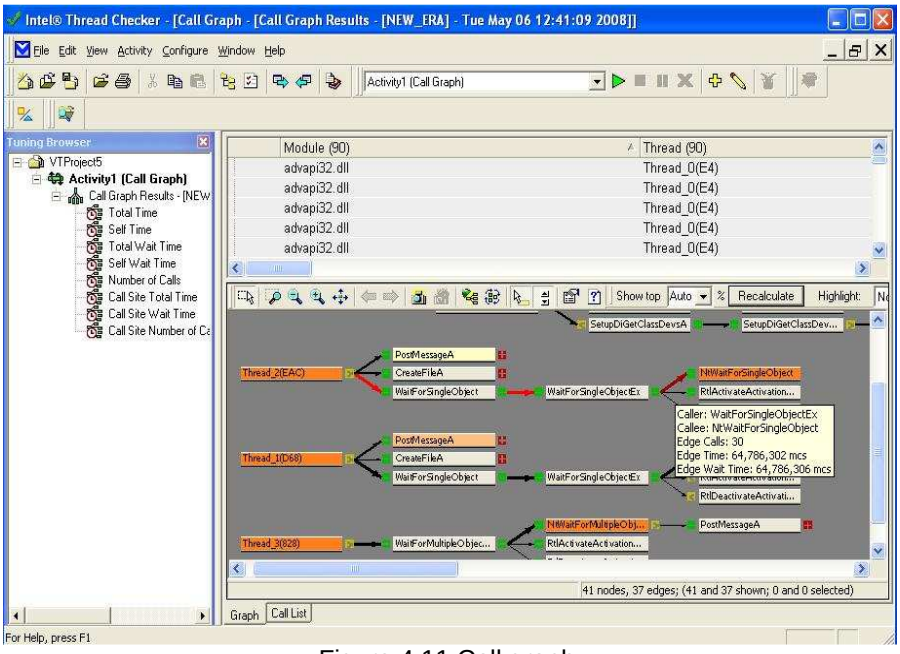


Figure 4.11 Call graph

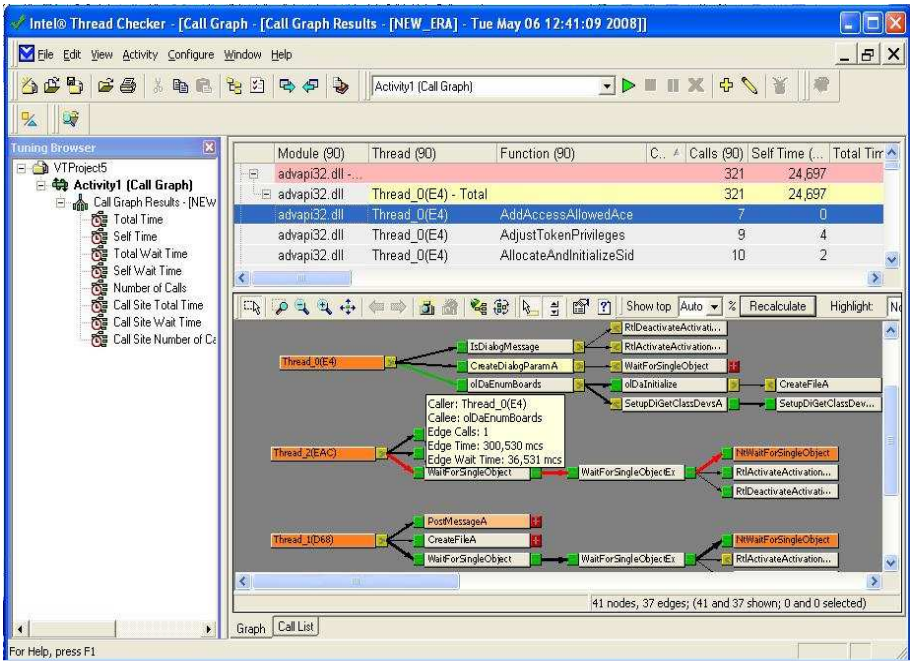


Figure 4.12 Call graph

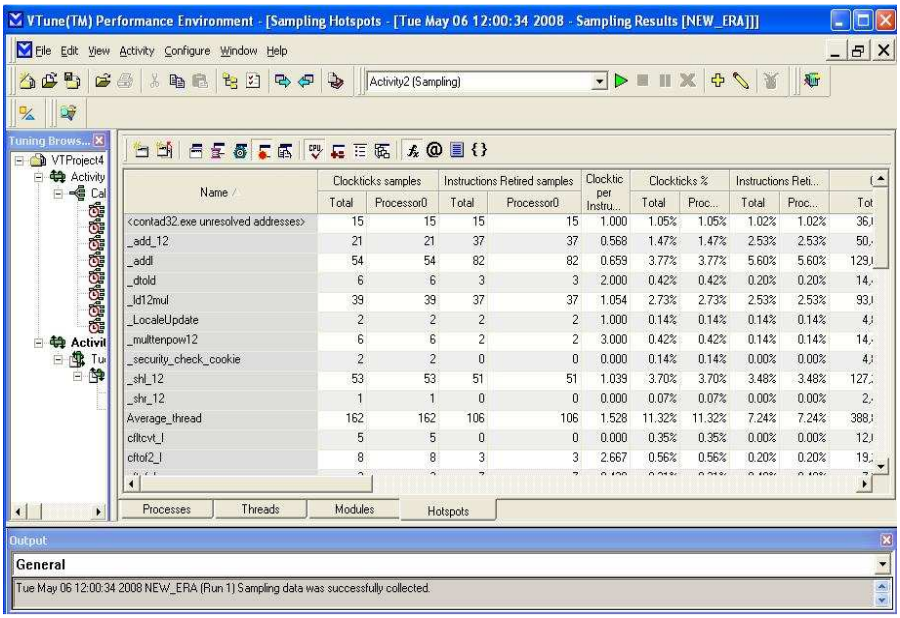


Figure 4.13 Sampling Hotspots

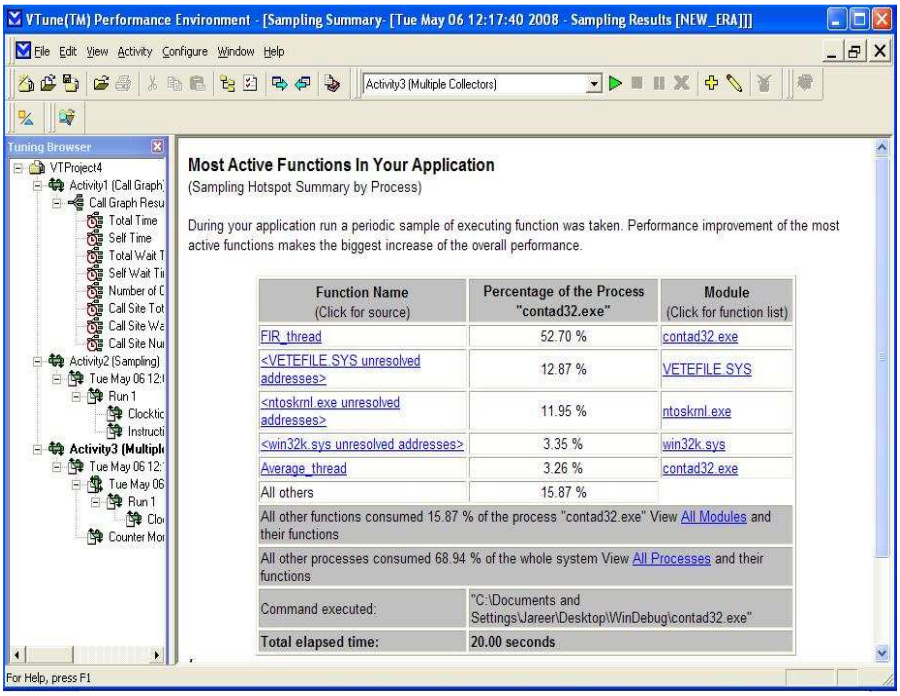


Figure 4.14 Process Sampling

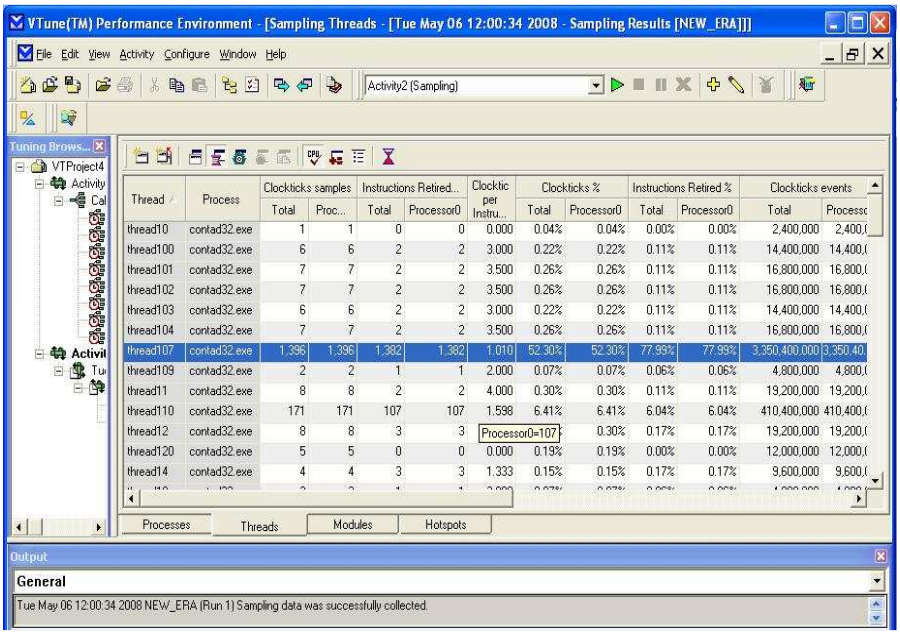


Figure 4.15 Threads

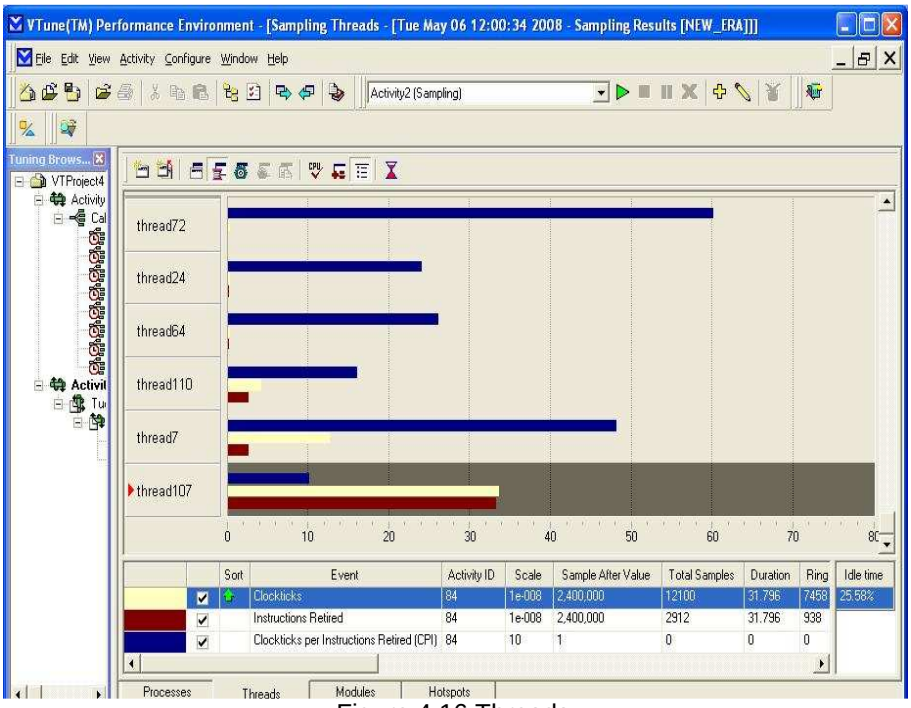


Figure 4.16 Threads

CHAPTER 5

EMBEDDED APPLICATION REAL-TIME ROAD PROFILER SYSTEM

An instrument widely used in transportation l engineering, is the high-speed inertial reference profiler. Profilers are used by most state and governmental agencies to measure the road surface profile to determine road roughness and classify road usability.

The road profiler system will be introduced, discussed and analyzed in this chapter, as part of designing real-time embedded applications. Algorithms and procedures used to calculate different statistics from road profile are also analyzed in this chapter such as the international roughness index (IRI), and texture estimation algorithm.

5.1 Problem Statement

Measuring road surface roughness (profiling) plays an important role in both constructing and maintaining roads. Road (pavement) profile is first computed from the raw data of the road section to be analyzed, and then special software packages such as ProVAL [59] are used to perform many types of profile analyses which is done off line.

Part of our research is to build a real-time road profiler that includes real-time solutions to many of the statistics used for the analysis of the computed profile. This system will have its own processing unit to carry all the computations.

5.2 System Description

The system to be designed will collect and analyze data regarding the road surface condition in order to specify road roughness and usability. The purpose of the system is to determine the road profile and texture measurements from the data collected by laser and accelerometer sensors.

The profiler is an instrument that is used to produce values related in a well-defined way to a road surface [55]. The following subsections will describe and discuss briefly the profiler system and the analytical algorithms implemented.

5.2.1. Profiler System

High-speed profilers (usually referred to as inertial profilers) combine reference elevation, height relative to the reference and longitudinal distance to produce the true road surface profile.

Most profilers measure profiles for the traveled wheel path. For each wheel path an accelerometer is used to find inertial reference defining the height of the accelerometer at that moment after double integrating the acceleration measurements. A laser sensor is then used to obtain readings representing the height of the road surface to the reference, and a distance encoder provides the longitudinal distance as illustrated in Figure 5.1.

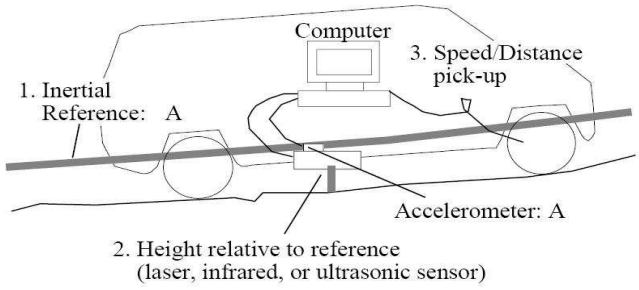


Figure 5.1 Road profiler system [55]

The road profile is reconstructed from laser and accelerometer readings according to the following equation.

$$p(t) = \iint a(t)dtdt - H(t) \tag{5-1}$$

Where

a(t) is the acceleration,

H(t) is the height measured by the laser sensor.

In Figure 5.2, a block diagram of the road profile computation is shown.

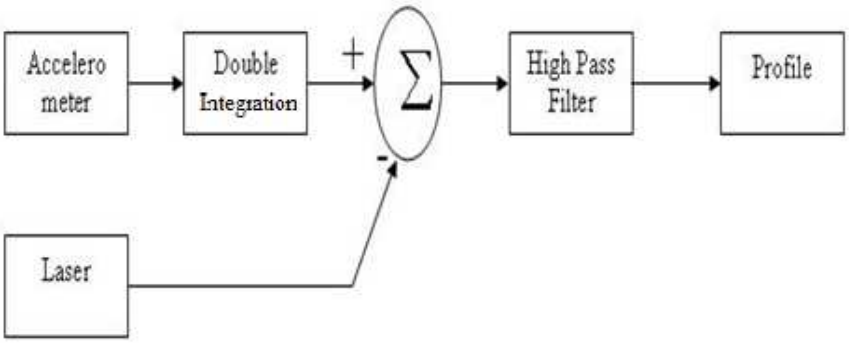


Figure 5.2 Road profile computation block diagram

A laser sensor installed as part of profiler systems shown in Figure 5.1 is used to collect texture data and provide an estimate of the amount of texture found in pavement. Figure 5.3 is an illustration of a road surface and the types of information that can be obtained using a profiler system.

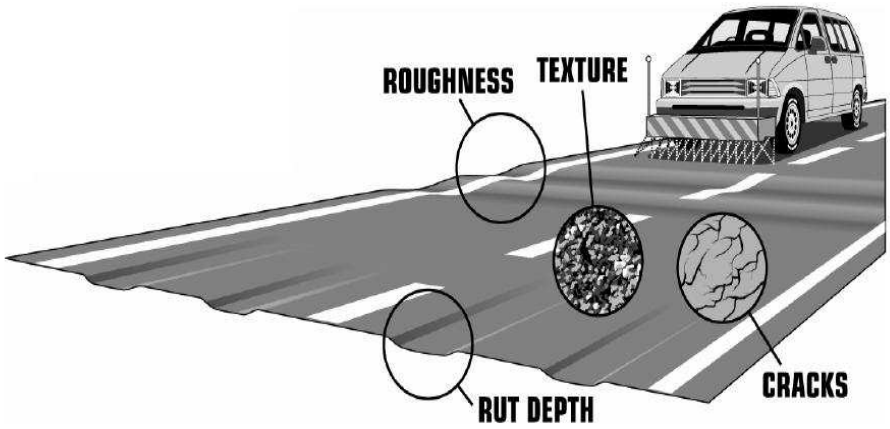


Figure 5.3 Profiler system

A high-pass filter is used to remove the effect of long wavelengths (low frequency components) on the profile. These wavelengths represent the underlying grade and overall road curvature and are more difficult to measure with inertial profilers with the current configuration.

There is also a distance sensor to measure the distance traveled. An optical sensor is often used to determine the start and end of specific road sections to be measured. The data is collected from all sensors via data acquisition system simultaneously sampling all sensors (six for dual profile systems) with sampling rate of 4 kHz or more.

5.2.2. Texture Estimation

Road surface texture is deviations from a planar surface that affects the tire interaction. Pavement texture is divided into three categories micro-texture with wavelengths from 0 mm up to 0.5 mm, macro-texture with wavelengths from 0.5 mm up to 50 mm, and mega-texture (roads) with wavelengths from 50 mm up to 500 mm.

ASTM E 1845 [57] standardizes the calculation of pavement texture from laser readings which represent the measured profile of the pavement macro-texture.

In order to compute the mean profile depth (MPD), the measured profile is divided into segments each having a base-length of 100 mm (3.9 in.). The slope, if any, of each segment is suppressed by subtracting a linear regression of the segment. The segment is further divided into two equal lengths of 50 mm (2 in.) segments and the height of the highest peak in each half segment is determined. The difference between that height and the average level of the segment is calculated. The average value of these differences for all segments making up the measured profile is reported as the MPD. See Figure 5.4.

For a profile system that computes both texture and profile, the texture is computed from laser sensor readings for both left and right wheel paths that are part of the road profiler system as discussed in the previous subsection. It should be noted, that currently, only the system developed in this research has a system with this capability for real-time operations.

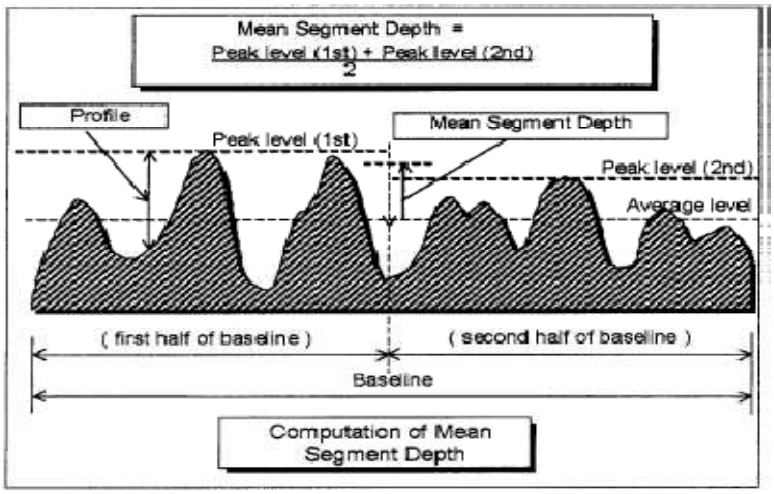


Figure 5.4 Mean segment depth computation

5.2.3. International Roughness Index

The most widely used statistic used for measuring roughness of pavements is the International Roughness Index or IRI. IRI summarizes the roughness qualities that impact vehicle response. IRI is computed from a longitudinal profile measurement using a quarter-car simulation (Figures 5.5 and 5.6) at a simulation speed of 80 km/h (50 mph) using equation (5-2) below.

$$IRI = \frac{1}{L}\int_0^{L/V} \left|\dot{z}_s - \dot{z}_u\right| dt \qquad (5\text{-}2)$$

where

$\dot{z}_s$: Spring mass velocity

$\dot{z}_u$: Unspring mass velocity

L : Profile Length

V : Speed, (80 km/h 50 mph)

IRI value is reported in either meters per kilometer (m/km) or inches per mile (in. /mile) (m/ km = 63.36 in./mile.) When calculating IRI, the quarter car parameters for masses, damping

and stiffness must be set to represent the Golden Car. The parameters (normalized to sprung mass $m_s = 1$) for the Golden Car are:

$c_s = 6.0$, $k_t = 653.0$, $k_s = 63.3$, and $m_u/m_s = 0.15$; where m_u is the un-sprung mass

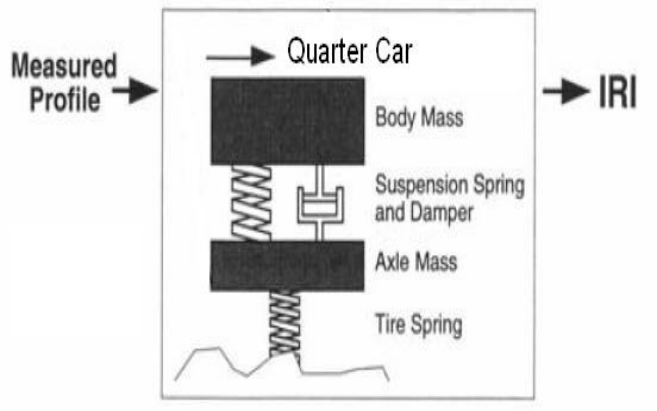


Figure 5.5 IRI computation [58]

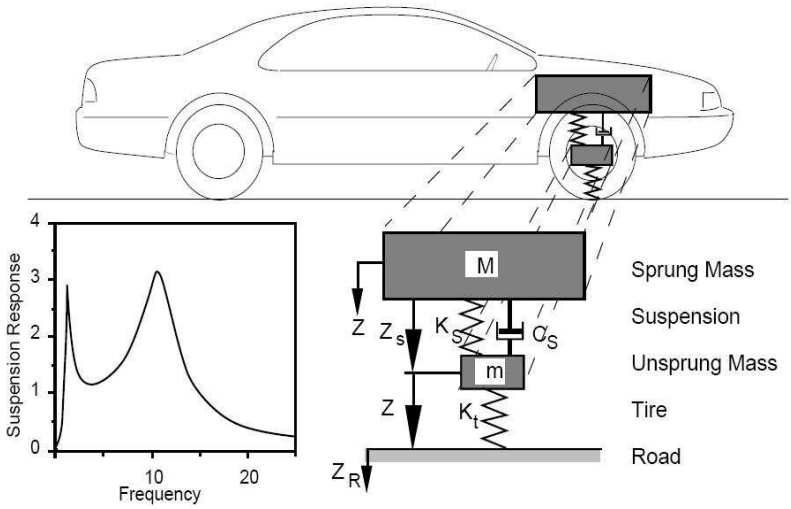


Figure 5.6 Quarter-car representation

5.2.4. Bump Detection Algorithm

The bump detection algorithm to be implemented in this system is known as template analysis procedure (TAP) which was developed in [54]. Although there are several methods used by government and state agencies, the TAP method is very comprehensive and more challenging for real-time applications.

The main concept of TAP method is to perform a cross-correlation between the road profile and 8-different bump templates of variable lengths.

The bump templates generated have lengths between one and eight meters. The road profile is cross correlated with the bump templates, while dips will not be generated since the cross correlation value between the templates and a dip will be negative. The absolute value of the cross correlation results is then compared to a threshold and if a value found to exceed the threshold level a bump (for originally positive correlation) or a dip (for negative correlation) will be recorded in an output report.

5.2.5. Hardware Requirements and Sensor description

The main hardware sensors and boards to be used in this research to perform the road surface analytical tasks including the profiler are:

- Laser sensor (Figure 5.7): To measure distance between the vehicle and the road surface which will be used for texture analysis as well as construct road profile for ride quality analysis and bump detection.
- Accelerometer (Figure 5.8): To measure the vertical acceleration of the vehicle's body.
- Distance encoder (Figure 5.9): To measure the distance traveled in order to specify the location of any deficiency in the road surface. The distance encoder used will be attached to the vehicle's wheel hub. GPS system or any other non-contact distance encoder.

- Optical sensor (Figure 5.10): To reference the start of road section to be scanned and tested. This is a photoelectric sensor that generates a high luminance beam spot to detect color differences or metallic surfaces. A mirror or a metallic tape will be installed at the beginning of the road section to be scanned. Once the photoelectric sensor detects the metallic surface it will generate a pulse which will be interpreted as start signal where the data analysis will start from that point.
- Data acquisition card (Figure 5.11): To digitize the analog data collected from the sensors.
- Processing unit (Laptop or embedded board): Use a multi-core based processing unit.

Figure 5.12 shows portable a profiler box after construction, while Figure 5.13 shows how the profiler box is attached to a vehicle.

Figure 5.7 SLS 5000 Laser Sensor

Figure 5.8 Accelerometer

Figure 5.9 Distance encoder

Figure 5.10 Optical sensor

Figure 5.11 Data Acquisition board DT9816

Figure 5.12 Portable Profiler Instrument Module

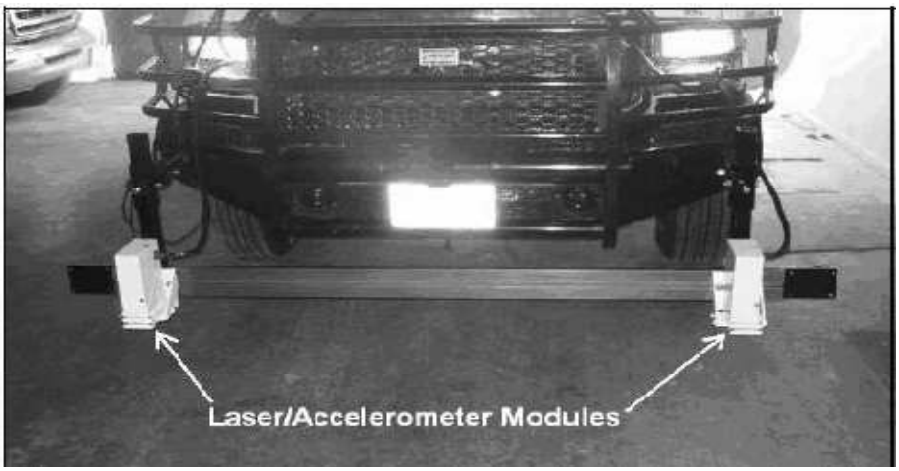


Figure 5.13 Road profiler system Installed in a Vehicle

5.2.6 Data Acquisition Requirements

The main requirement to operate the road profiler/ texture system is to collect data for any road section with the minimal distraction to the traffic, which means that the profiler vehicle should operate in a speed range of 40- 60 mph (58- 88 ft/sec). The shortest road section to be tested is about 0.1 of a mile (528 ft), which takes about 6- 9 seconds to cover this distance with the operating driving speed. The data from all 6 sensors are to be collected simultaneously via data acquisition system in order to compute the road profile values correctly.

The minimum sampling rate required to construct road profile from sensor readings is 4 kHz; while for accurate texture estimation, laser readings should be sampled with at least 24 kHz (63 kHz for systems using the texture laser). So, in order to implement the road profiler/ texture system sampling rate required is 24 kHz or more, since all sensor readings should be sampled with the same speed for consistency.

For the required operating driving speed and 24 kHz sampling rate(63 kHz for systems using the texture laser), this system should be able to collect and process at least 144000 (378000) samples every second per sensor. The data acquisition system as well as the

embedded processor used and the available memory unit should all be able to collect, process, display, and store large amounts of data in real-time without missing any single reading.

5.3 Algorithms and Analysis Methods

This section will concentrate on the analysis of several of the algorithms and tasks that are related to road profiling. In The first subsection profile reconstruction algorithm will be discussed pointing to the effect of filters used in the reconstruction process. The second subsection analyzes and simulates quarter car model along with the IRI filter model and the effect of different texture levels in road surface in both models. The last subsection is dedicated to the texture estimation and introduces a solution for estimating texture using data collected from low speed laser sensors. It also addresses the texture issue currently a problem with single point lasers.

5.3.1. Filter Design and Profile Reconstruction

5.3.1.1 Infinite Impulse Response (IIR) Filters

IIR filters are one type of filter categorized in terms of their impulse responses. The other type is the Finite Impulse response (FIR). The IIR filter can be implemented as either analog filters, using electronic components like op amps, or digital filters, by applying mathematical operations on a sampled signal.

Digital IIR filters design are implemented in two phases, the first phase is to design the filter as an analog IIR filter then, in phase two, convert the analog filter to a digital filter form by applying discretization techniques like bilinear transform. There are many types of IIR filters, most known types are Chebyshev, Butterworth, and Elliptic filters.

IIR filters are usually described pole-zero transfer function which represents the order of the filter. The following equation is the for the general transfer function.

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_L z^{-L}}{1 + a_1 z^{-1} + \cdots + a_L z^{-L}}$$

Where L is the filter's order, while a and b are the filter coefficients.

IIR filter is the filter that is used in the profile computation algorithm. As last step in profile computation, a high pass typically an IIR filter is used to filter the profile to attenuate all wavelengths of 200 ft and higher, or low frequency components of the generated signal (profile). Profilers typically use either use 3^{rd}- or 4^{th}- order Butterworth filters in the process of computing road profile from the raw data.

One of the main IIR filters disadvantages is the nonlinear phase shifts that occur to the filtered signal with respect to the original signal before filtering; this disadvantage often requires an extra processing step to remove the nonlinear phase shift when measuring short sections. This step involves the use of a zero-phase filter. Zero-phase filters, or reverse filters as known in the profiler design terminology, consist of filtering a signal using the designed filter. The signal is then reversed and applied to the filter in the reverse direction to obtain a filtered signal.

In the case of FIR filters, it is possible to design linear phase filters that, when applied to data it simply delays the output by a fixed number of samples. But for IIR filters, however, the phase distortion is usually highly nonlinear. This process is easily accomplished with languages such as Matlab, but the process must be implemented in C for our real-time application. The filfilt Matlab function uses the information in the signal at points before and after the current point, in essence looking into the future, to eliminate phase distortion. [62]

After filtering the data in the forward direction, filfilt reverses the filtered sequence and runs it back through the filter. The result has the following characteristics:

- Zero-phase distortion.
- A filter transfer function, which equals the squared magnitude of the original filter transfer function.
- A filter order that is double the order of the filter specified by filter coefficients.

The Matlab code in Figure 5.14 is an example that illustrates the effect of a zero-phase filter and compares the results of filtering a signal with and without using zero-phase filtering. In the example, a 4^{th} order low-pass Butterworth is designed with a cutoff frequency of 10 Hz and

applied to a signal consisting of two sine waves with different frequencies (3 and 40 Hz). The signal is filtered using the zero-phase filter applied by the filfilt Matlab function; also it is filtered using filter Matlab function that filters the given signal once (forward filtering only) the results are shown in Figure 5.15 where it is clear that the signal filtered with zero-phase filter has the same phase shift of the original (unfiltered) signal.

```
t = 0:0.01:1; % Sampling Frequency Fs = 100 Hz
x = sin(2*pi*t*3)+.25*sin(2*pi*t*40);
% Design IIR Butterworth low-pass filter with cutoff frequency = 20 Hz
% Cutoff frequency is normalized by Fs
[b,a]=butter(4,20*.01,'low');
% Filter x with zero-phase filter
YL = filtfilt(b,a,x);
% Filter x without applying zero-phase filter
Y = filter(b,a,x);
plot(t,x,t,YL,'r',t,Y,'k')
legend('Original Signal',...
'Filtered Signal with filtfilt',...
        'Filtered Signal with filter')
```

Figure 5.14 Matlab code for Zero-Phase filter effect

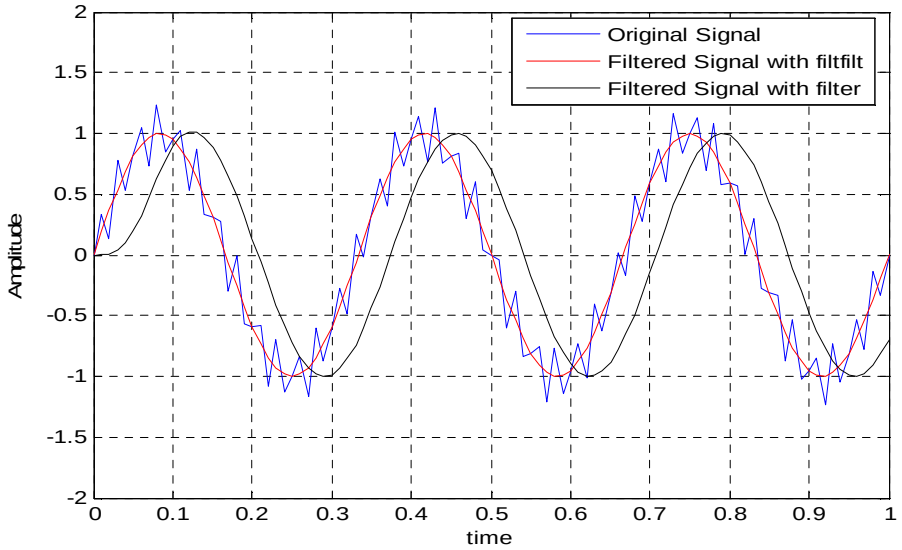


Figure 5.15 Comparison of the effect of zero-phase filter

5.3.1. 2 Comparison of the 3rd and 4th -order High Pass IIR Filter

A comparison between 3rd and 4th -order IIR Butterworth filters (typically used by profilers) and its effect on reconstructing the reference profile will be shown here. The first step is to compare the step response of both filters which is shown in Figure 5.16. The 3rd order filter appears to have settling time shorter than the 4th order filter. Applying the zero-phase filtering method as in Figure 5.17, the step response of both filters is identical, which means that both filter orders are candidates for usage in profile computation.

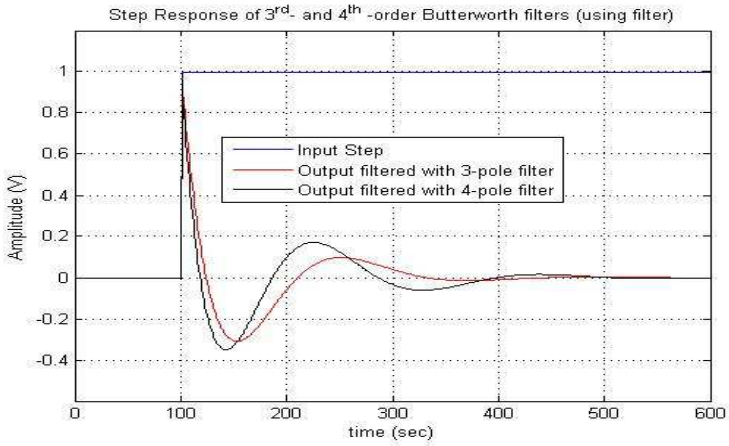


Figure 5.16 Step Response Comparison of 3rd- and 4th-order Filters

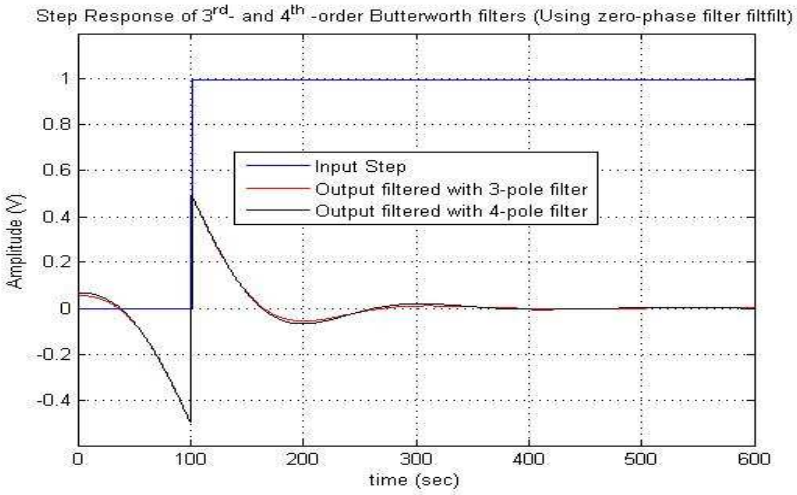


Figure 5.17 Step Response Comparison of 3rd- and 4th-order Zero-Phase Filters

5.3.1. 3 Filtering Effect on Profile Computation

After building a road Profiler, it has to pass a certification test set by the state department of transportation. A certified profiler can then be used to collect profiles from any road section for the purpose of checking the road conditions as well as using it by contractors in the process of building new roads. The certification is done on a selected road section. Each section is at least a tenth of a mile (528 ft) in length. The section profiles are first constructed using traditional profiling instruments such as rod and level or dipstick. These profiles are called reference profiles, since they will be used as a reference to be compared to the profiles collected by the road profiler under certification. Figure 5.18 shows a reference profile collected from a section of SH47 state highway in Bryan Texas. The reference profile was collected over a distance of 1140 ft as part of a TXDOT project described in [63]. Within this section, a test segment of 528-ft was established. This distance provides sufficient lead-in and lead-out intervals for verifying the accuracy of inertial profile measurements based on the requirements given in Tex-1001S.

The reference profile will be a collection of height measurements. In order to view different types of the profile features it should be filtered. The filter type used to filter the reference profile should be the same filter that is used by the high-speed profile under certification. The same wavelengths are filtered from both profile sets; which will be adequate to compare the reference profile to the profiles collected by the high-speed profiler in order to check whether this road profiler passes certification or not. For a profiler to pass certification, it must collect profile data from the test section. The profiler will make several runs (at least three) that follow the same wheel path using software packages such as ProVAL to compare those profiles with the reference profile collected previously. The profiles collected from the multiple runs should be identical with the reference profile with a very high accuracy where the point to point comparison among these runs should have a difference of less than 20 mils (thousandth

of an inch); also the IRI values of all runs should have repeatability not more than 3.0 inches/ miles.

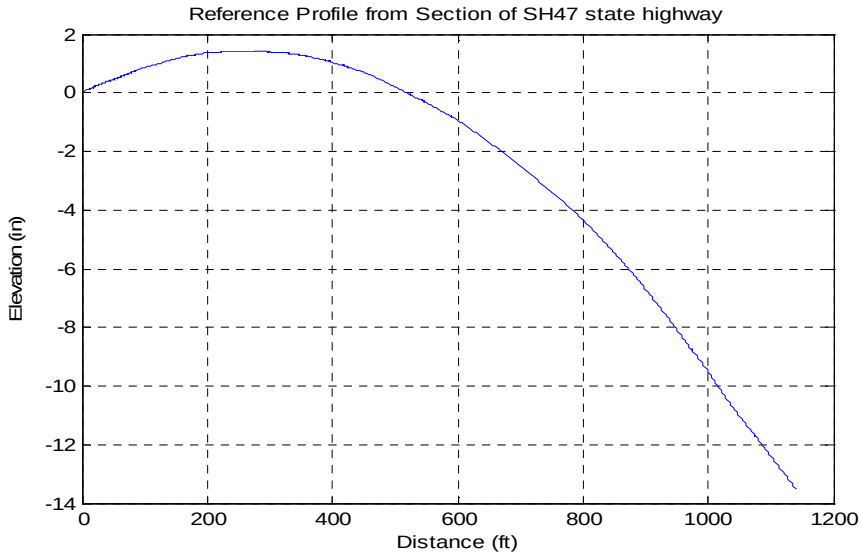


Figure 5.18 Reference Profiler

The filter used in most cases is a 3rd or 4th order Butterworth filter. The filter order differs among road profilers. The filter used must be delivered to the certification committee in order to filter the reference profile with the same filter used by the profiler. The filtered profile should have a zero reference which represents the smooth road surface. Figure 5.19 shows the filtered reference profile obtained from highway SH47 with varying filter orders. The filter is used once to filter only in forward direction for all four cases. The figure shows that the 1st order filter generates a profile that is much different than the other three filter orders. The 2nd order filter generates a profile with shift in its values, and does not have a zero reference. The 3rd and 4th order filters generate almost identical profiles as concluded in the previous section.

Using the same filters with the zero-phase filter option is shown in Figure 5.20. Note the much better results for all filter orders tested. From Figure 5.20 it is clear that 2nd-, 3rd-, and 4th-order filters generate almost identical profiles with slight variations in certain intervals, but the 1st

order filter, although performs much better using only forward filtering, and generates a profile that is offset from the other (desired) profile results.

One of the profile measurements obtained by the road profiler under certification is shown in Figure 5.21 of the tested segment. The figure compares the results of using different filter order (3rd-, and 4th- orders) which both produced a certifiable profile when compared with the reference profile.

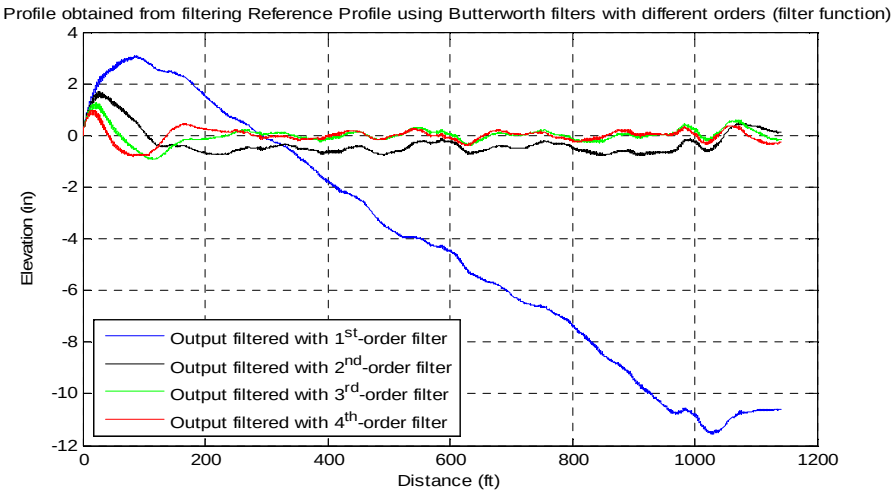


Figure 5.19 Effect of Filtering Reference Profile with Different order

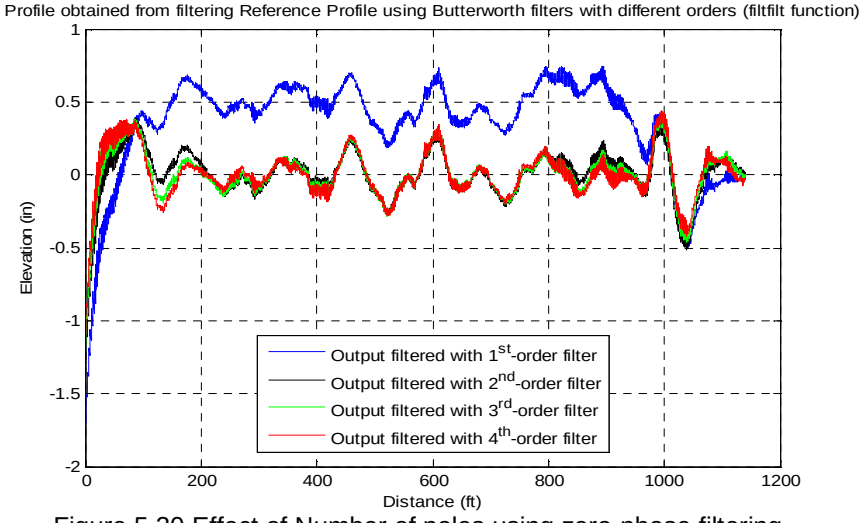


Figure 5.20 Effect of Number of poles using zero-phase filtering

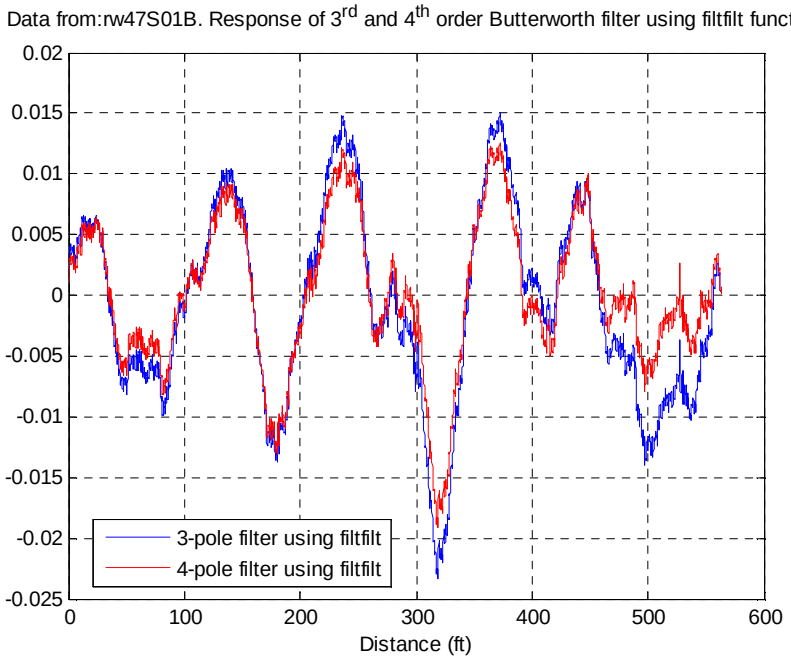


Figure 5.21 Repeatability of Inertial Profile Measurements on SH47 Test Segment

5.3.2. IRI Filter and Quarter Car Modeling

The International Roughness Index or IRI is used by state and federal agencies for indicating the level of roughness or smoothness of pavements. IRI is typically computed after profile data has been collected. During this research, a real-time version is planned for the embedded analyzer.

It has been found that IRI is adversely affected by texture. During the research, a model of IRI was developed and then used for determining why texture, which consists of very high frequencies or short wavelengths would have an adverse effect on IRI.

A model for the quarter-car (and the IRI filter) was established after analyzing Figure 5.6 and using equation (5-2) as a set of state-space equations (5-3).

Analyzing the state-space equations transfer function for every part of the model was constructed representing the displacement of the sprung and un-sprung masses as well as their velocity. Using the state-space and the transfer functions, a Simulink model (Figure 5.22) was

created. The model simulates and generates the response of every part of the quarter car model.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u, \qquad y = \mathbf{C}\mathbf{x} + \mathbf{D}u$$

$$\mathbf{x} = \begin{bmatrix} z_s \\ \dot{z}_s \\ z_u \\ \dot{z}_u \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -k_2 & -c & k_2 & c \\ 0 & 0 & 1 & 0 \\ k_2/\mu & c/\mu & -(k_1+k_2)/\mu & -c/\mu \end{bmatrix} \qquad (5\text{-}3)$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & k_1/\mu \end{bmatrix}^T$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \mathbf{D} = \mathbf{0}$$

where

$c = c_s / m_s = 6.0, \quad k_1 = k_t / m_s = 653,$

$\mu = m_u / m_s = 0.15, \quad k_2 = k_s / m_s = 63.3$

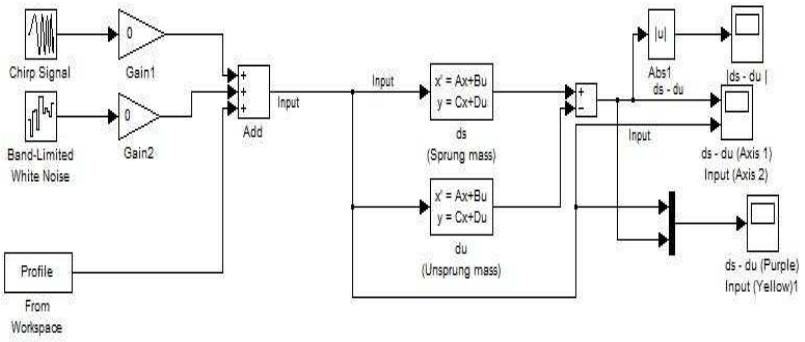


Figure 5.22 Quarter-car/ Suspension system Simulink model

To test the Simulink model, first a sine wave input (chirp signal) with variable frequency (between 0- 100 Hz) was used as an input to represent an input road profile (to simulate driving over a road constructed from sine waves with multiple frequencies) to determine the frequency response of the system. Figure 5.23 illustrates the output of the model which is the frequency response of the system. The response is found to be similar to the behavior of the quarter car model.

Another test is to check the effect of noise on the model. A band-limited white noise was used as an input; since texture on road surfaces is often characterized as a band limited white noise. Figure 5.24 shows the effect of white noise only on the system; it appears that the noise can generate an effect equal to its input amplitude. Using sine wave contaminated with white noise shows that the additive white noise affects the quarter-car model as well as the IRI filter and increases the amplitude (gain) of the frequency response for all frequencies.

Profile was collected and used in the previous subsection shown in Figure 5.21 as an input to the Simulink model to test its effect on the model. The result is shown in Figure 5.26. The profile shows a minor effect on the model, leading to the conclusion that the section tested has a smooth surface and in a very good condition. Computing the IRI over this section also confirms this conclusion as the IRI values were very low (average IRI = 38.6 inches/ mile) as reported in[63].

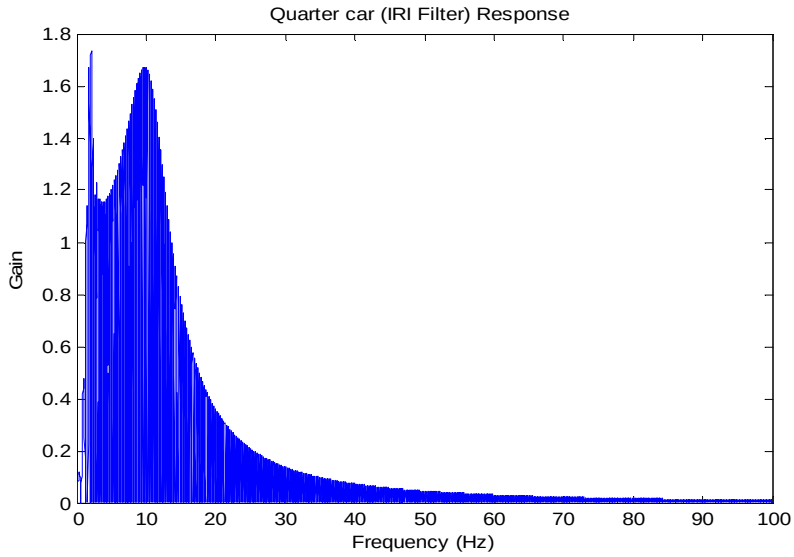


Figure 5.23 Frequency Response from Quarter-car Simulink Model (Input is sinewave with variable frequency)

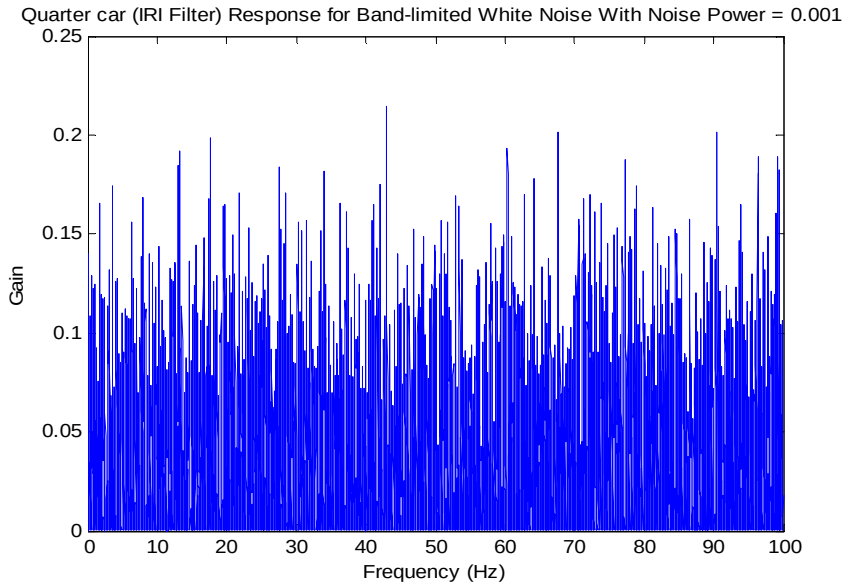


Figure 5.24 Quarter-car Response for Band-limited White Noise Input (AMplitude = 0.25, Power = 0.001)

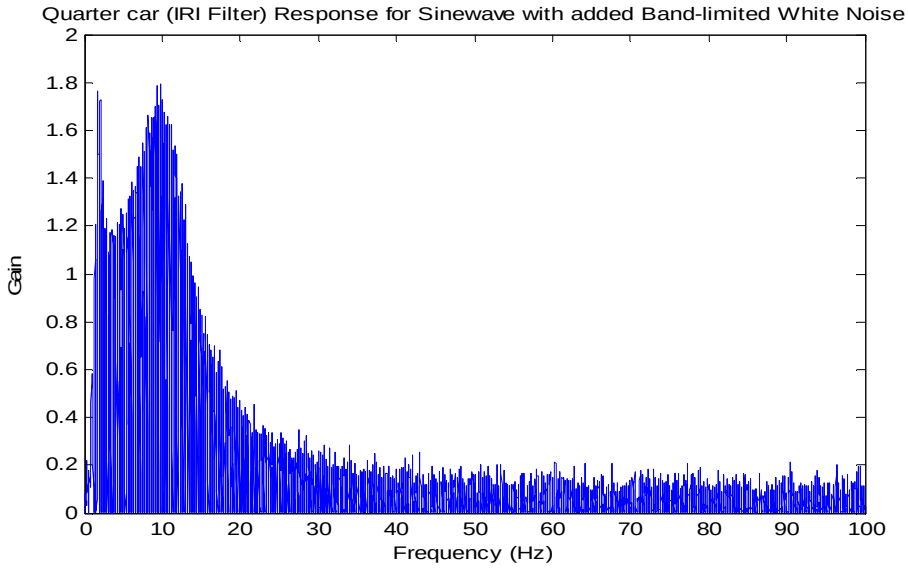


Figure 5.25 Quarter car Response for Sine Wave with added Band-limited White Noise

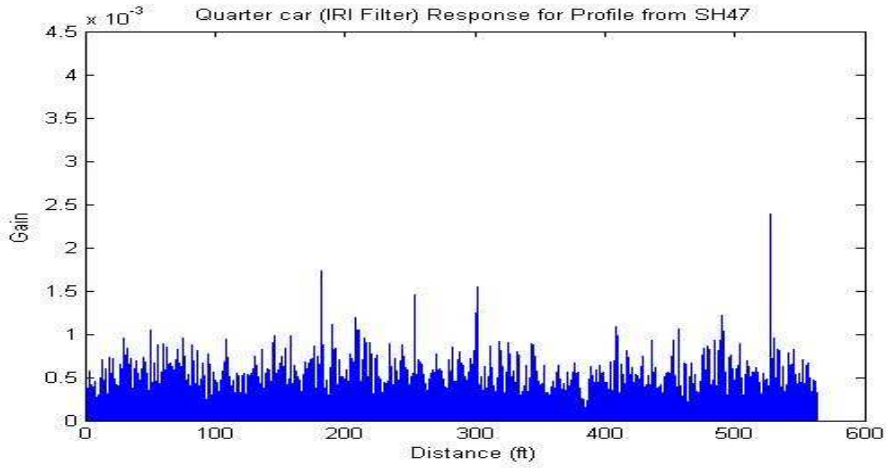


Figure 5.26 Profile Effect on Quarter-car Model

Figures 5.27 to 5.31 show the frequency response for the sprung, un-sprung masses after simulating the quarter-car model. This response is known as the IRI filter (quarter-car filter.) Figure 5.27 represents the sprung mass (car body) frequency response, which can be interpreted as the response of the car body to the road profile. The figure shows that maximum gain (response) occurs at around frequency of 1.0 Hz. The un-sprung mass (tire) frequency response is shown in Figure 5.28 where the maximum gain is obtained when hitting a wave with

around 10 Hz of frequency. The IRI filter response is the difference between those two responses which is shown in Figure 5.29. Finding the frequency response of the IRI filter in terms of wavelength is illustrated in Figure 5.30 in which signals wavelengths of 2.24 and 15.87 meters/cycle has the most affect on the quarter car system among all other wavelengths.

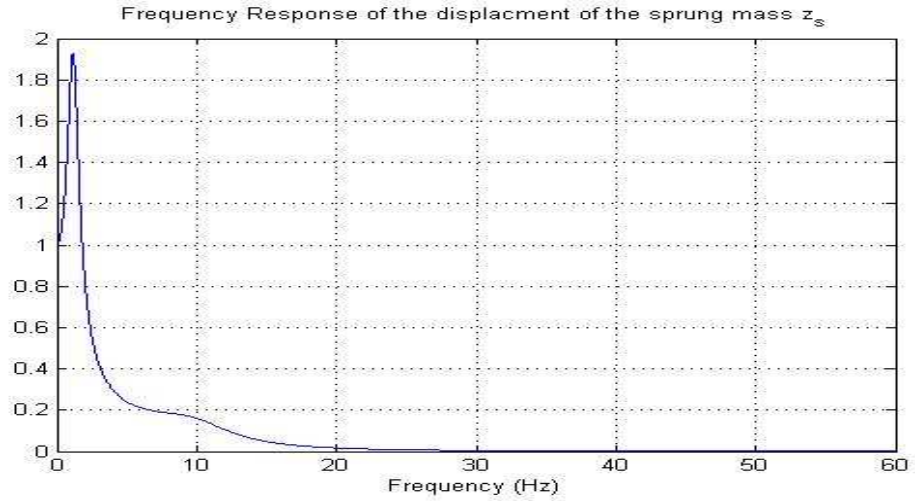


Figure 5.27 Sprung mass (car body) response

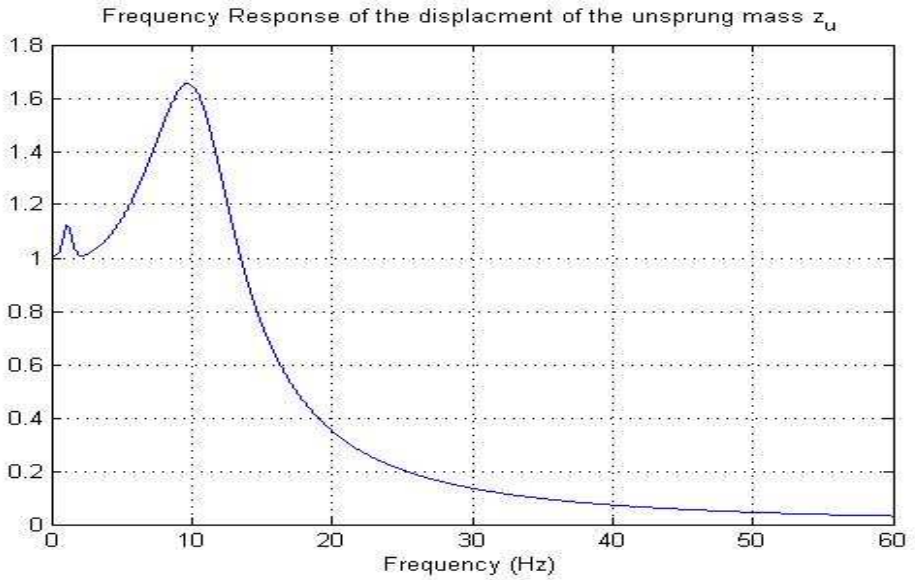


Figure 5.28 Un-sprung mass response

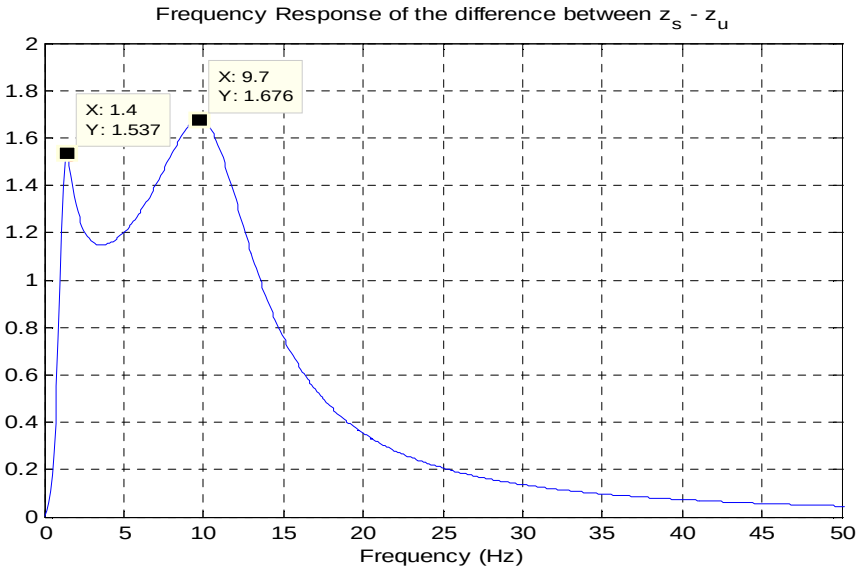


Figure 5.29 Quarter-car (IRI filter) Frequency Response (Frequency Domain)

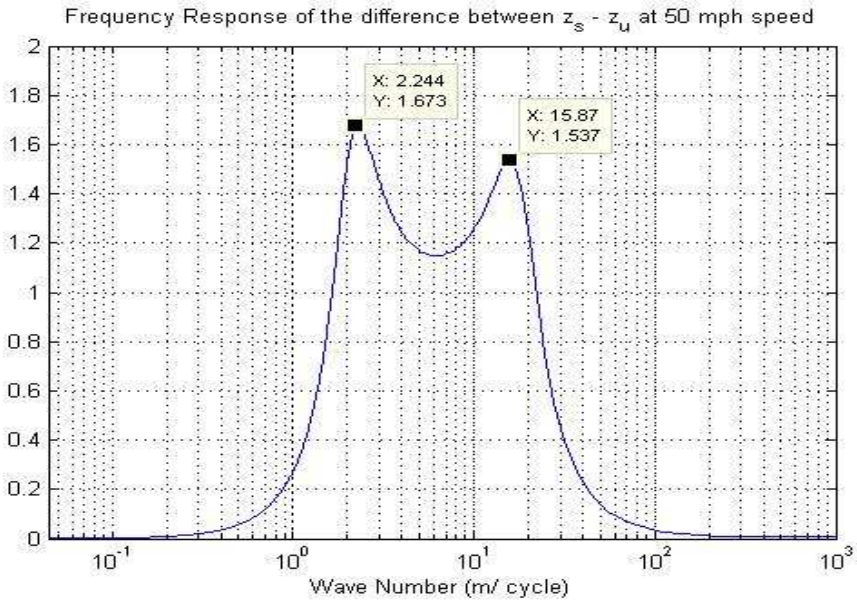


Figure 5.30 Quarter-car filter response (Wavelength)

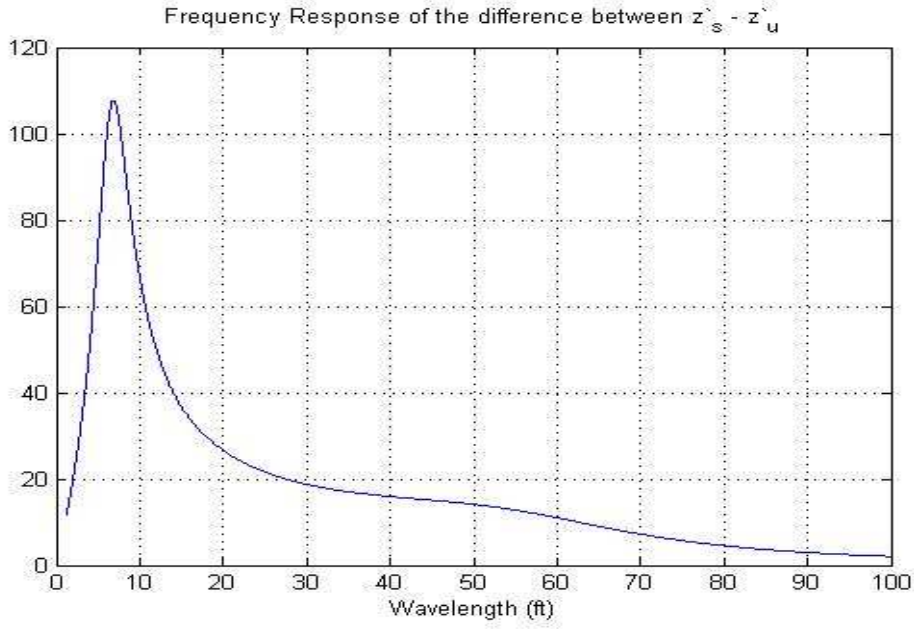


Figure 5.31 Deference of the response of the sprung un-sprung velocities

5.3.3. Texture Effect on IRI Filter

From the frequency response of the IRI filter in Figure 5.32 it appears that the texture range has a very small effect on the IRI filter. But several laboratory studies as well as IRI values for pavements with textured surfaces showed that texture content of the pavement causes an increase in IRI values by 8 to 10 in/mile. [60]

This means that texture has an effect on IRI values although not visible in the frequency response.

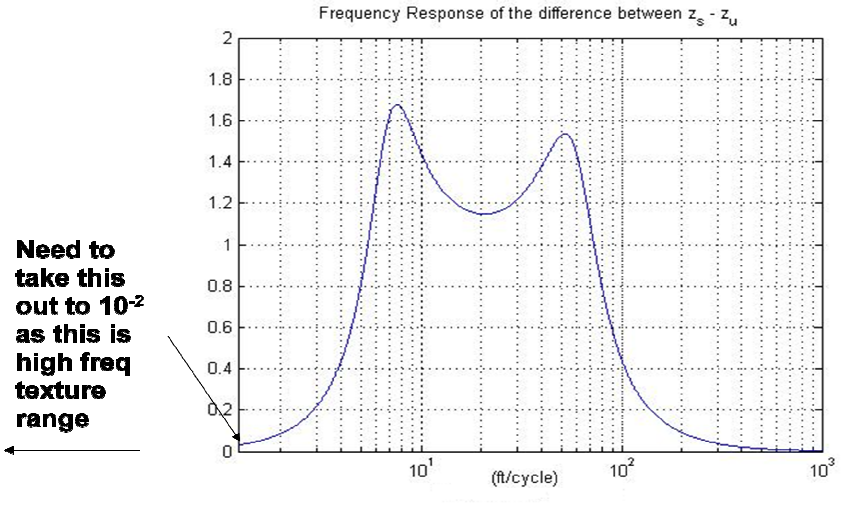


Figure 5.32 Texture Effect Range in IRI Filter (Quarter-car) Response

As part of texture effect laboratory study [61], several texture specimens (i.e. Figure 5.34) with different texture grades were made. The specimens were installed on base plate (shown in Figure 5.33). The plate is then rotated by a motor at a speed to emulate driving speed on road and where profile data were collected to test the effect of different texture grades. The profile of the base plate was also collected.

Figure 5.33 Base Plate Used to Hold Texture Specimens

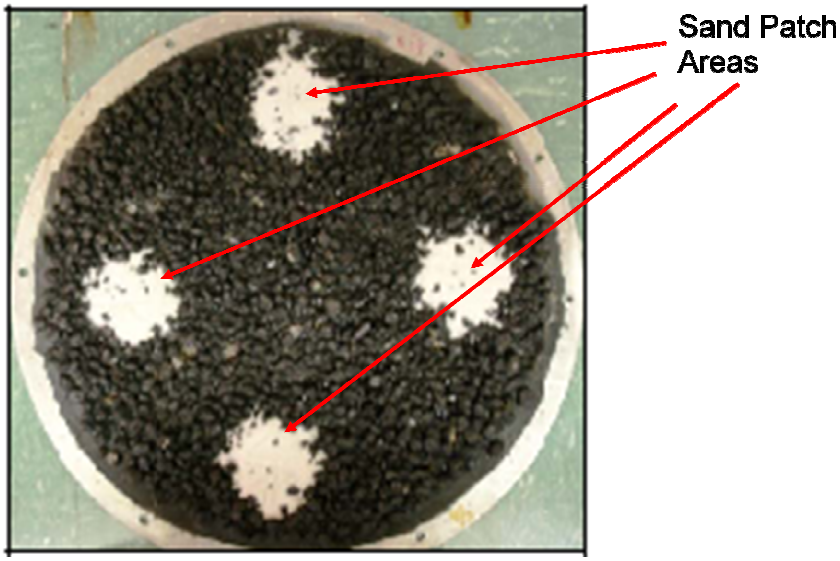


Figure 5.34 Plate with Texture

The IRI values for all profiles were computed and found that the base plate has an IRI of 3.0 in/ mile while texture specimens generate an IRI values in the range between 10.0- 36.0 in/ mile. This effect is illustrated in Figure 5.35 where it shows not only that the IRI value is increased by texture content, but also that the power spectrum is increased in terms of adding (or multiplying) by a constant while preserving the original (base plate) shape of the power spectrum. In other words, the base plate power spectrum was magnified by adding (or multiplying by) a constant value to all frequencies found in the base plate power spectrum. This can lead to the conclusion that texture can be a white noise. (Notice that the frequencies up to 20 cycles/ inch were filtered out using wobble filter to eliminate the effect of wobbling generated by the rotational speed.)

The findings agree with the general conclusion of pavement engineers that the texture can often be characterized as band-limited white noise. The spectral density of white noise has a flat power spectral density. That is, the signal contains equal power within a fixed bandwidth

at any center frequency. White noise is defined with a flat spectrum over a defined frequency band.

In this subsection we have focused on analyzing the texture effect on IRI and how can the effect be minimized. Current methods to minimize affects of texture on IRI are based around new lasers with wider footprints, line lasers, etc., and using different bridging filters.

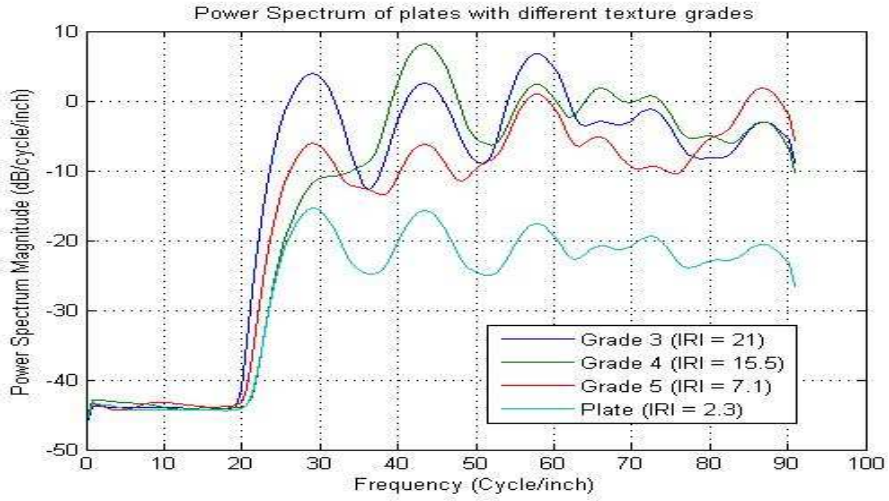


Figure 5.35 Power Spectrum Comparison for different Texture Grades

As further analysis white noise was generated using Matlab with a magnitude and standard deviation equal to that of one of the texture specimens used (the grade 4 over 3 texture specimen) and then added to the base plate signal to resemble the texture specimen. The signal (profile) generated is shown in Figure 5.36 and was compared to the texture specimen select for the analysis and to the base plate signal, which shows that white noise generate signal is identical with the texture specimen profile signal. Also the IRI value for the generated signal is close to the textured plate IRI value and much higher that the IRI value of the base plate, as stated in Table 5.1.

A father analysis is made by comparing the power spectrum of all three signals to determine the degree of resemblance between white noise and textured signals where such resemblance will prove that texture can be classified as white noise. Looking at the power

spectrum of all three signals as illustrated in Figure 5.37, it can be noticed that although the power spectrum for the white noise-based signal is in the range of the textured plate, but it is clearly shown that white noise doesn't preserve the shape of the original power spectrum of the base plate. Instead it has a flat power spectrum without any distinctive features that can be found in the base plate and the textured plate power spectrums.

A solution for texture effect on IRI values is to define ranges, or classes, for the IRI values for every texture grade. Then according to texture grade of certain road section its IRI value is to be compared to the class range in belongs to in order to determine the road's condition and serviceability status.

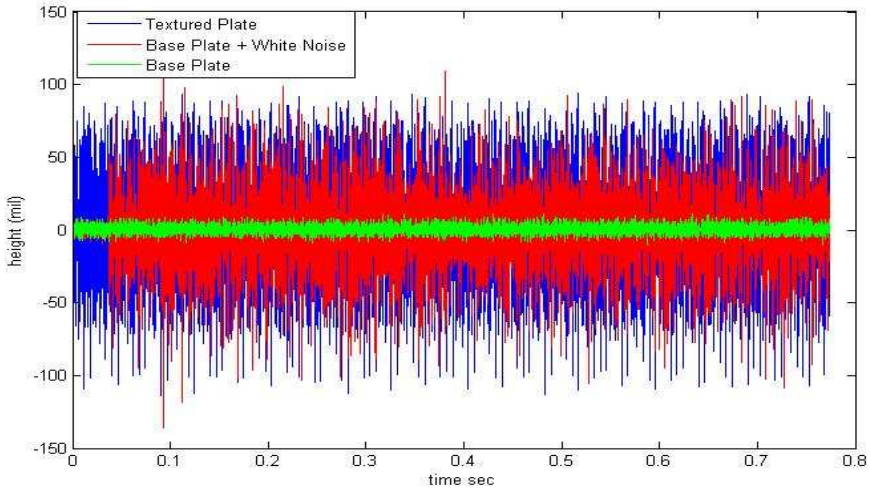


Figure 5.36 Comparison of the Addition of White Noise and Texture to Base Plate

Table 5.1 IRI value Comparison

Profile	IRI (in/mi)
Plate	3.0
Textured Plate	12.7
Plate + White Noise	12.8

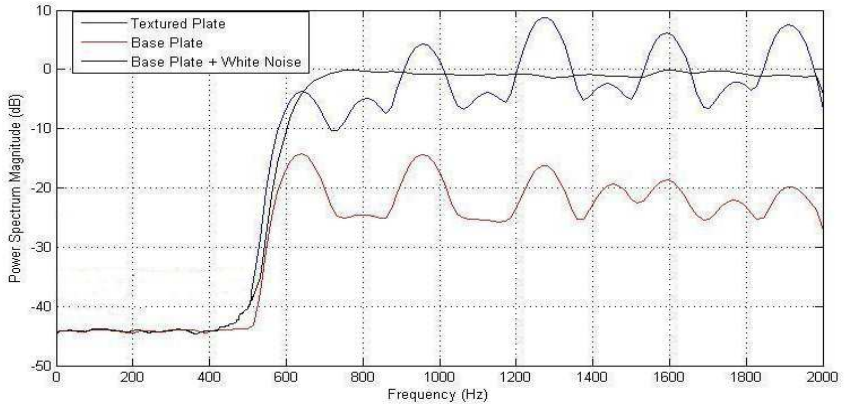


Figure 5.37 Power Spectrum for Base Plate, Plate with White Noise, and Textured Plate

5.3.4. Texture estimation Using Low speed laser sensor

Most road profilers until recently use the low speed sensor such as the SLS 5000 which has sampling rate of up to 16000 samples per second (16 KHz) as part of the profiler system to collect data regarding road profiles. In order to compute the MPD value to estimate texture contents of the road, other specialized laser sensors are used which are known for their high resolution readings and of their fast sampling rates. As an example the LMI texture laser sensor that can provide 62.5 KHz. These laser sensors also are more expensive than regular profiling laser sensors (texture sensors cost at least three times more than the cost of profiling sensors.) This means that for any road section, the road surface is measured twice one for profiling and the other to collect data for texture analysis.

For this reason and as part of this research we explored the possibility of estimating the texture contents of the road surface based on the profiling sensors.

The following subsections will introduce and discuss the results obtained from using the SLS 5000 laser sensor to measure texture, and the correction equation concluded by this research to estimate the correct MPD values based on the sampling rate and vehicle's speed.

5.3.4.1 SLS 5000 laser sensor test

The SLS 5000 laser sensor was tested first to check whether it can be used to collect details of pavement surface. A profiler was used to collect data with low speed (10 mph) and high sampling rate (8 kHz) the system was able to show considerable details regarding pavement section measured as illustrated in Figure 5.38 and the laser reading for this section as in Figure 5.39.

Figure 5.38 Tinning from section measured from SH130

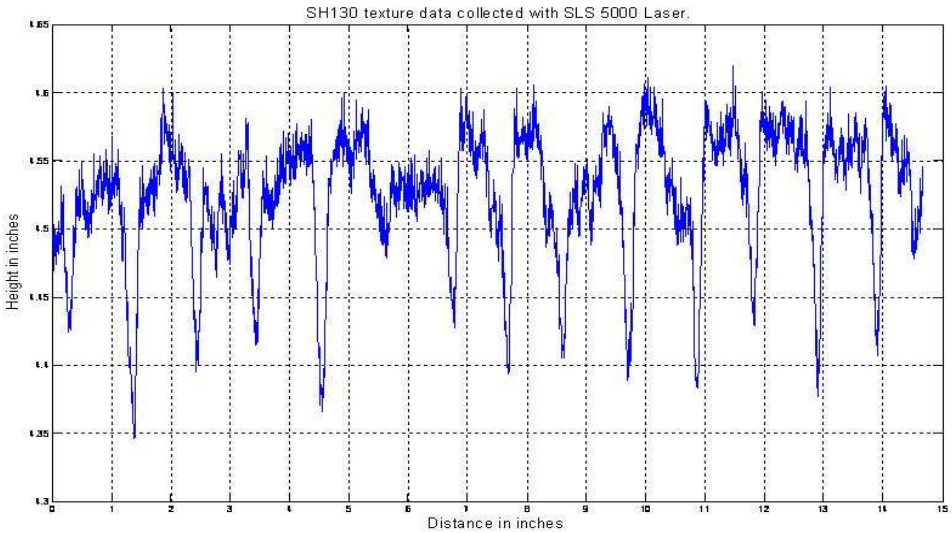


Figure 5.39 Collected data showing tinning in measured section

5.3.4.2 MPD Frequency response

The frequency response of the MPD algorithm was obtained by generating sinusoidal waves with different periods (lengths) representing that fits in a single segment length (100mm) used for computing MPD using following formula.

$$X = A\sin\left(2 * \pi * \frac{time}{Period}\right) \ , \ Where\ A = 474, and\ Period\ = 1{:}500$$

In order to generate a single sine wave that covers a single 100mm segment the period value has to be 164, when comparing to distance encoder readings from road profiler running at 10mph. The value of amplitude (A) was set to be similar to a laser reading. Figure 5.40 illustrates having single sine wave in a 100 mm segment. Figure 5.41 shows the frequency response of the MPD algorithm which resembles filter system that passes waves with short periods (less than 400) and attenuates waves with higher period values. In other words it passes signals with high frequencies and attenuate low frequency waves. This means that MPD algorithm acts like a high-pass filter. Figure 5.41 shows the frequency response of the MPD algorithm which resembles a low-pass filter system.

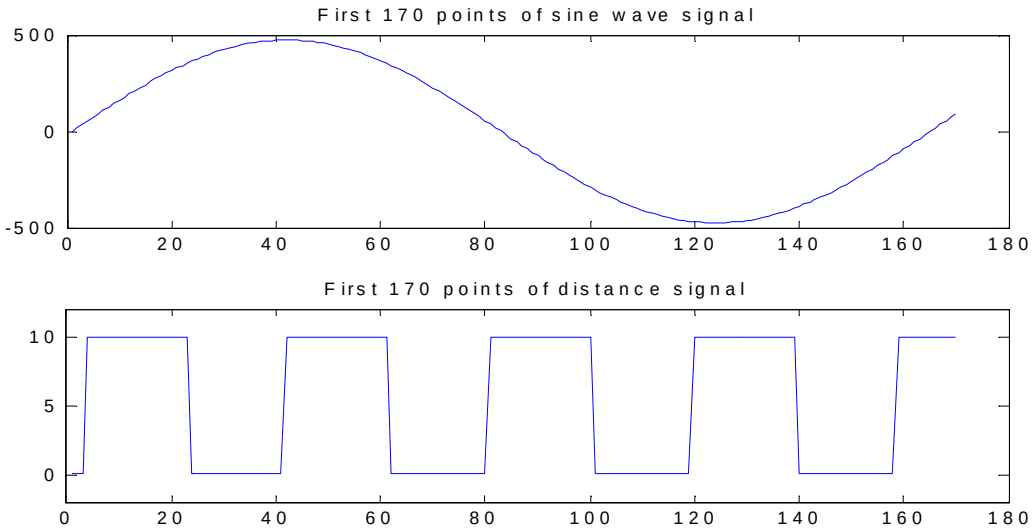


Figure 5.40 MPD Segment with Single Sine wave and Distance Encoder Reading for the same Segment

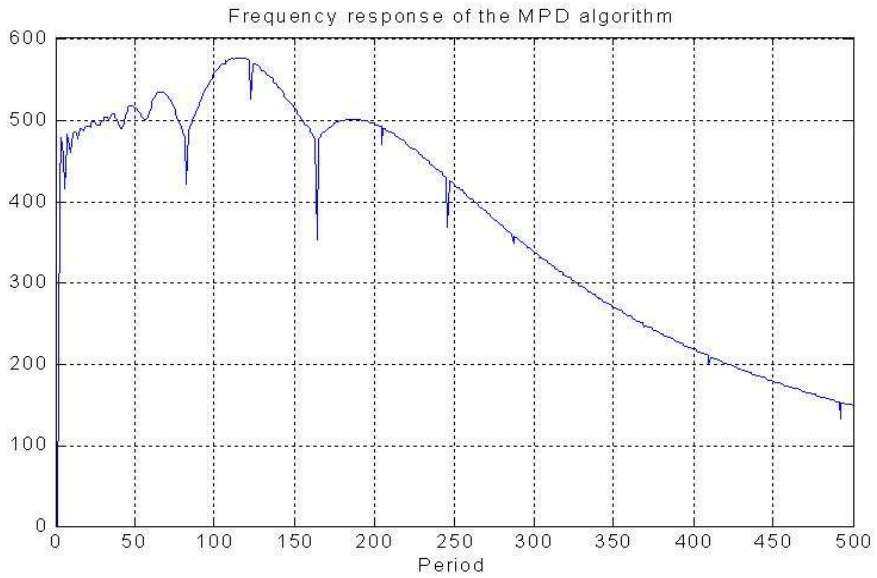


Figure 5.41 MPD frequency Response

5.3.4.3 Speed effect on texture measurements

After several data measurements sessions it was observed that the texture measurements are affected by the vehicles speed, in a way the higher the vehicle's speed the less details of the texture is collected which leads to unrealistic representation of the correct pavement texture . In order to overcome this obstacle extra measurement were done using the highest sampling rate that the data acquisition system can support accompanied with the fastest readings that the sensor can provide. Speeds used were between 10- 40 mph in an increment of 10 mph with repeat runs for every speed and driving over different types of pavement sections with different texture contents. Accordingly it was found that sampling rate of 24 kHz is adequate enough to collect data at high speed with reasonable amount of information that can provide a good estimate of the texture contents of the pavement.

It was also found that driving at speed of 10 mph with sampling rate around 8 kHz is adequate enough to provide correct texture measurement of the pavement. It was also found that a linear relation was noted between the MPD values and the speed. See Figure 5.41 for example of the linear relationship for measurements of MPD at TTI middle test section.

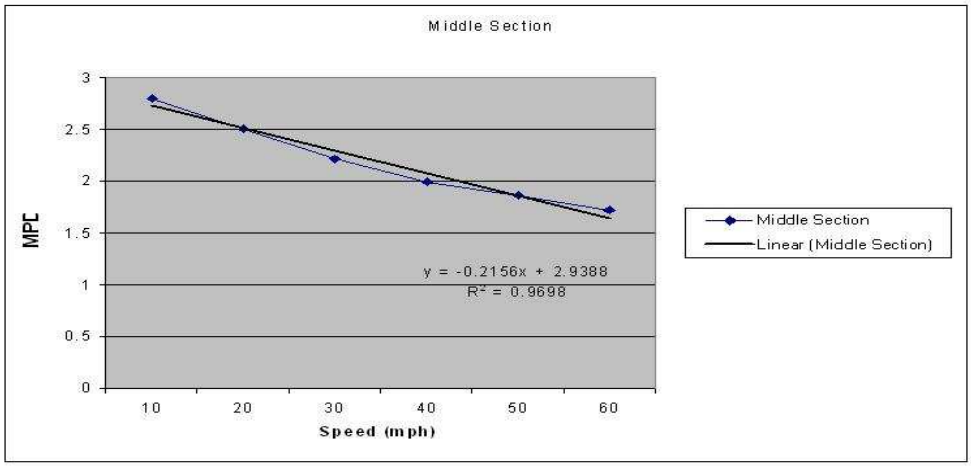


Figure 5.42 Relationship between MPD different driving speeds

From that regression was computed and a formula was obtained that provides a conversion factor of the texture measurements from high speeds to the reference 10 mph texture measurements.

$$MPD(10) = 0.37495 + SPD * -0.01209 + MPD(SPD) * 0.91093 \quad (5\text{-}4)$$

Where *SPD: is speed in mph*

5.3.4.4 Sand patch vs. MPD

A series of sand patch readings were made over the same sections in which the laser readings were taken. It was noted that there appeared to be a direct relation between these readings and the MPD readings at 10 mph.

ASTM E 1845 defines the sand patch estimate from MPD using equation (5-5)

$$ETD = 0.2 + 0.8 * MPD \quad (5\text{-}5)$$

From the measurements done we found that the sand patch estimate can be computed from the data collected using the SLS 5000 at 10 mph as in equation (5-6) below.

$$ETD = 0.3 + 0.8 * MPD \quad (5\text{-}6)$$

5.3.4.5 Results

Regression values obtained from comparing MPD measurements at different speeds compared to MPD at 10 mph, as well as regression between 10 mph MPD values compared to sand patch readings are defined so that MPD can be measured at any speed from 10 to 40 mph and the MPD at 10 mph can be estimated with R^2 of 96.6. The MPD at 10 mph can be used to predict the Sand Patch reading with an R^2 of 97.4 (Figures 5.43 and 5.44.)

Future work will be in finding a relationship of MPD values of measurements done at higher speeds, as well as constructing a formula that ties the speed of the vehicle, and the sampling rate to the reference MPD obtained from measurements done at 10 mph.

Also, another task to be done is to compare results obtained using the SLS 5000 sensor to results from texture laser sensor. Process texture data in real-time using dedicated embedded system. . It should be noted that only a small sample of texture sections were investigated. A much larger sample is needed before any final conclusions.

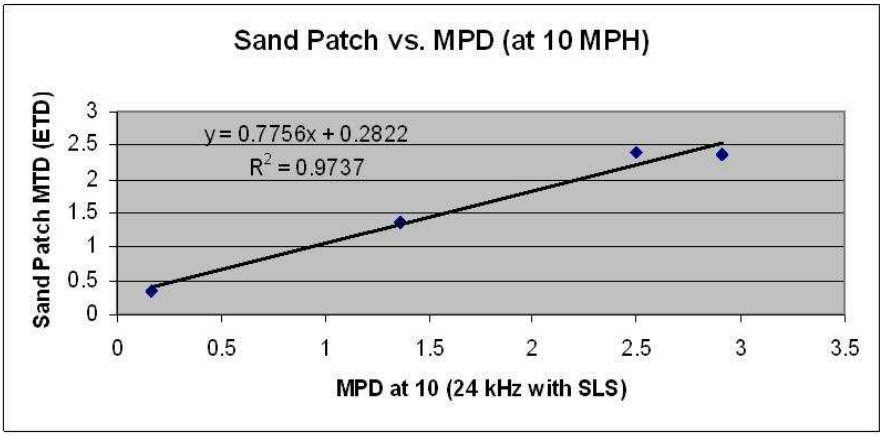


Figure 5.43 Regression of Sand-patch vs.MPD at 10 mph

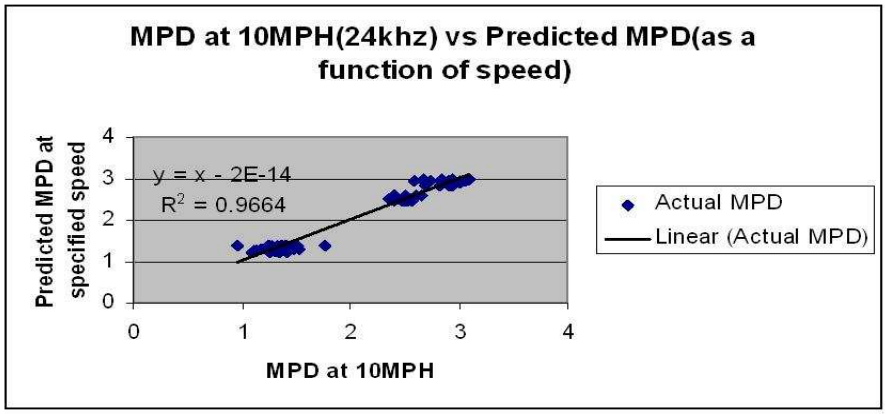


Figure 5.44 Regression of MPD at different speeds vs. MPD at 10 mph

CHAPTER 6

SYSTEM IMPLEMENTATIONS AND SAMPLE RESULTS

In this chapter, the design and implementation of three different profiling systems are discussed. Section 1 is for a real-time road profiler/ texture system which is a partial implementation of the real-time road surface analyzer system. Although this system shares the main hardware equipments required in the design of real-time road surface systems, and in general any profiler system, it differs from it by two factors. First, the processing unit used; where a system like that usually uses a low end embedded board that may have an old low speed processor such as the Diamond Athena II boards shown in [1], or one of the new embedded processors that come either as a single core or dual core processor like the EP80579 and the ATOM processors from Intel. Section 2 shows full design of the real-time road surface analyzer system.

The third section discusses another concept of profiling which uses the 3-D profiling system, where a 3-D image for the road surface can be reconstructed using laser and gyroscope reading. This system does not use any type cameras. Although this is a prototype system, it represents a new area in profiling where having 3-D representation of the road surface can provide important information to assess the road usability and other information needed for pavement maintenance considerations.

6.1 Design, Implementation, and Evaluation of Real-time Road Profiler/ Texture System

The following subsection will show and discuss the process used to model the real-time road profiler/ texture system based on UML profile for MARTE then the results from implementing and running such a system with the aid of ATOM and EP80579 based boards will be introduced.

6.1.1. Road Profiler/ Texture System Modeling

This subsection focuses on the development of a model that is used for measuring both profile and texture in real-time. The embedded system, include sensor hardware, and multi-core processors and is designed with a real-time UML extensions. The system is a first step in designing and building a fully integrated real-time system that implements profiler/ texture systems and other applications such as rutting, video, etc. We used a combination of UML and MARTE to model the system. We use MARTE only when UML has limited support.

The first step in modeling with UML is to provide a description of a system's behavior which is done with the aid of a use case diagram. Figure 6.1 illustrates the use case that describes the profiler/ texture system. The figure shows two actors. The road surface itself under test, and the system's operator that is responsible for driving and running the measurement vehicle. This use case is described by a series of events that occur regarding operating the profiler/ texture system. The road surface will be measured to determine the reference elevation from accelerometer sensor and height relative to the reference from laser sensor of the road surface. A combination of both data sets will be used to compute profile, while the texture will be estimated using the height readings alone.

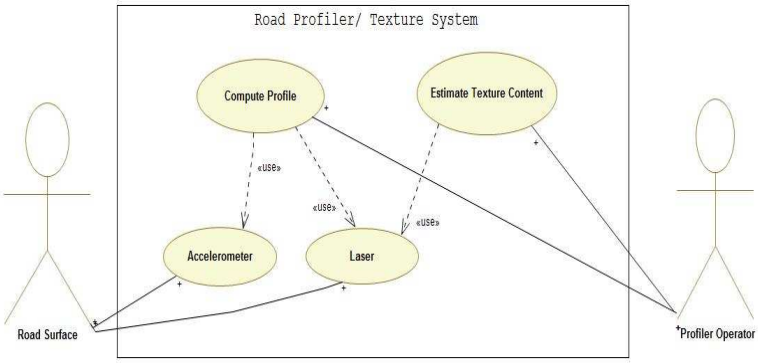


Figure 6.1 Road Profiler Use Case

The activity diagram is used to show workflows (flowchart) in a step by step manner for the activities and actions, with support for choice, iteration and concurrency. The state (activity) diagram is another diagram of the UML standard that shows the step-by-step workflows of the

activities and actions. The activity diagram of the system here is shown in Figure . The main activity this system carries is to continuously collect sensor data. The data is then distributed between two computational tasks processed concurrently for profile construction and texture estimation. The concurrent operations are represented in the activity diagram with the aid of fork, join states. The fork pseudo-state is a connector that branches a single input transition into multiple outgoing transitions to different states that will be activated concurrently. The join pseudo-state joins together multiple incoming transitions into a single transition. Once the data collection is over, and the system is ready to stop working, all data points will be saved in output files for offline analysis and archiving processes.

Next, the state machine is introduced (Figure 6.4). There are two states. The idle/ pre-section state is where the system starts running but doesn't perform any computation. The real section state is where the system collects and processes the data. UML models parallelism in two ways. First, all objects are considered to be parallel entities. Second, a single object entity exhibits itself in a concurrent behavior. This means that the object's state-machine is specified as a set of concurrent components. The real section state is supposed to perform three operations separated by the dotted lines. The first operation is to collect the sensor readings, while the second operation is to perform texture analysis once enough data is obtained (4 inch worth of data). The third and final operation is to compute profile for every 1 inch. Those three operations are intended to be performed concurrently as stated in the state machine.

In order to model the Tolapai embedded processor we created new stereotypes. One stereotype is for the QuickAssist technology and the other is for the Tolapai processor itself adding the capability of customizing them towards individual application domains. The new stereotypes will appear in the diagrams as <<QuickAssist>> and <<Tolapai>> respectively. Figure shows the <<Tolapai>> stereotype, which extends three of the MARTE meta-classes namely hw_processor to represent the IA-32 processor in the Tolapai, the hwI_O for the I/O

controller part, the hwMMU for the memory management unit. It also extends the <<QuickAssist>> stereotype.

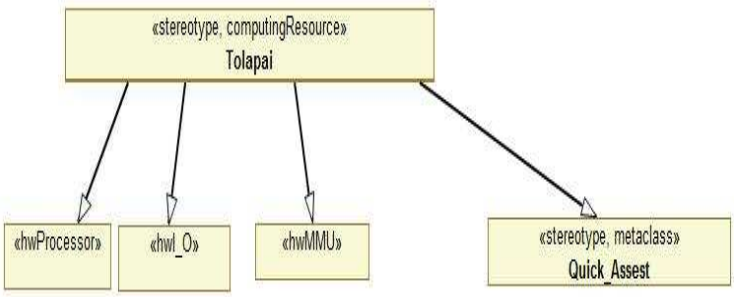


Figure 6.2 The Tolapai Stereotype

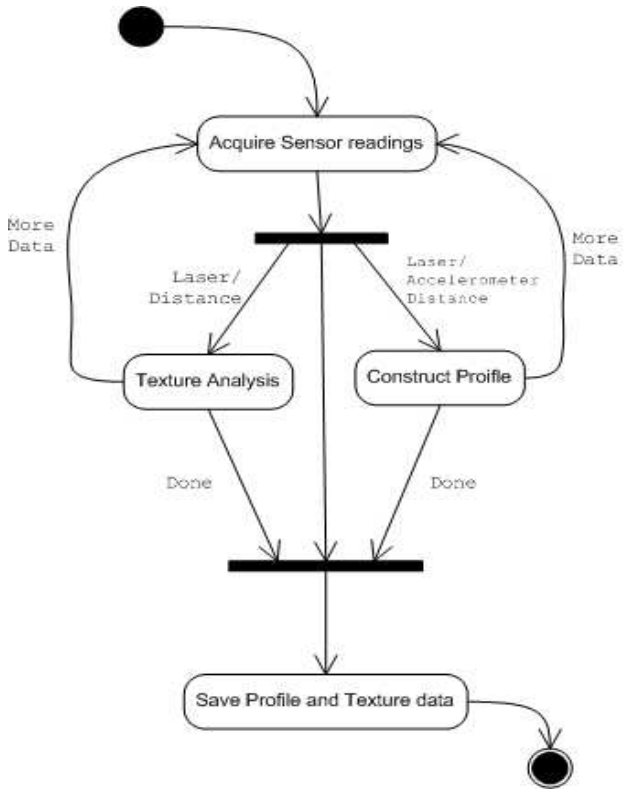


Figure 6.3 Activity Diagram

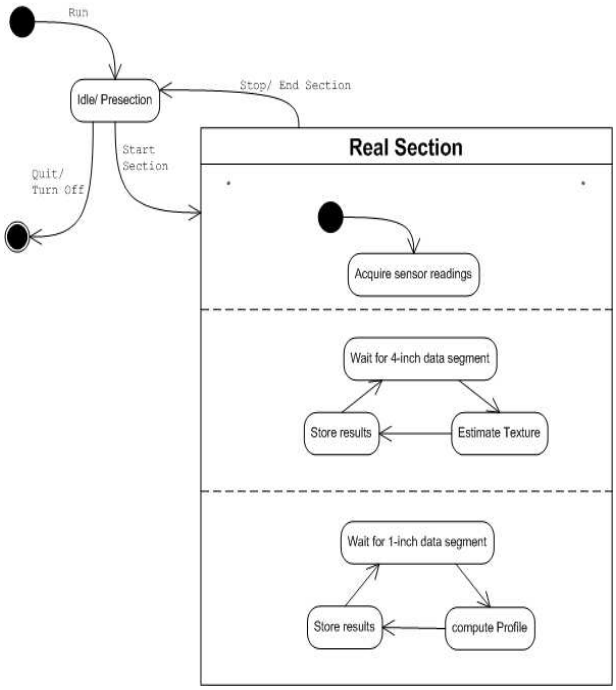


Figure 6.4 State-Machine

Figure 6.5 illustrates the HRM model of the road profile/ texture system. Using the stereotypes defined by MARTE for different hardware components plus the newly created stereotype for the Tolapai processor we were able to model and specify the hardware devices and controllers to build such a system. The software functions and methods to be used are modeled with the aid of SRM profile as in Figure 6.6.

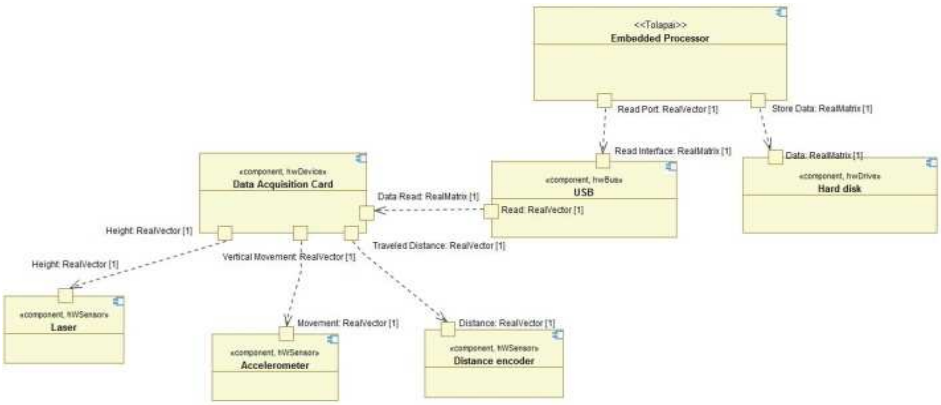


Figure 6.5 Road Profiler HRM Model

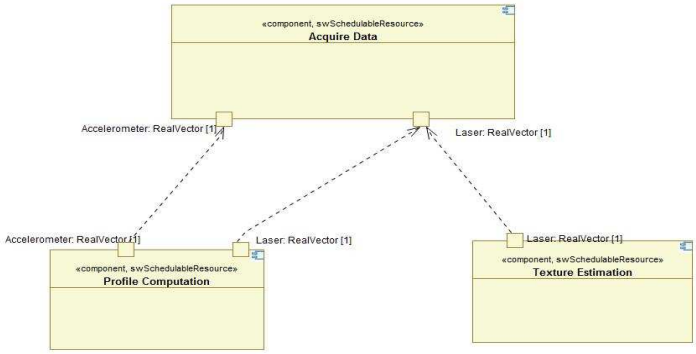


Figure 6.6 Road Profiler SRM Model

6.1.2. Applicability of Tolapai for the Road Profiler/ Texture System

The profiler program was next used with the Tolapai development system to simulate real-time data collection. A portable instrument package used for obtaining raw sensor data for a given wheel path, Figure 6.7, was used to obtain sensor data from a typical road section. The desired plan was to use the Tolapai inside the instrument module, processing, sending the computed profile texture and other pavement performance characteristics via a network connection to a client computer. Using this data the multi-threaded profiler program was run on the Tolapai development system to compute profile for one wheel path, simulating the real-time data measurement process. The computed results matched with the real-time measurements using the current measurement system, Figure 6.8 represents the road profile while Figure 6.9 shows the MPD values for texture estimate.

This profiler is tested in real-time mode using a single core ATOM based system for certification purposes. The results obtained from that system performed well and passed the certification according to the Texas Department of Transportation (TXDOT) requirements.

Figure 6.7 Portable Profiler Instrument Module

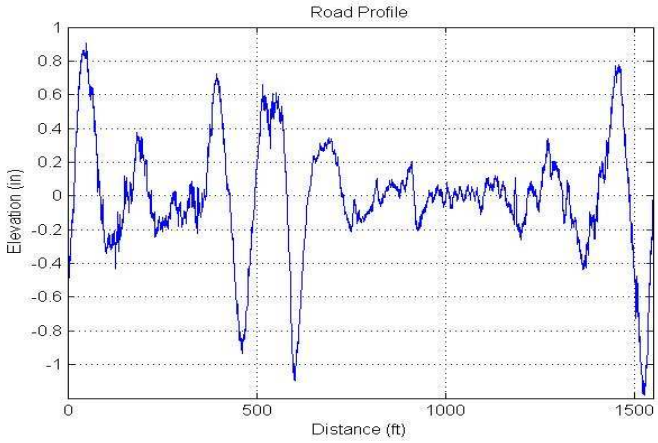


Figure 6.8 Road Profile measurement for tested section

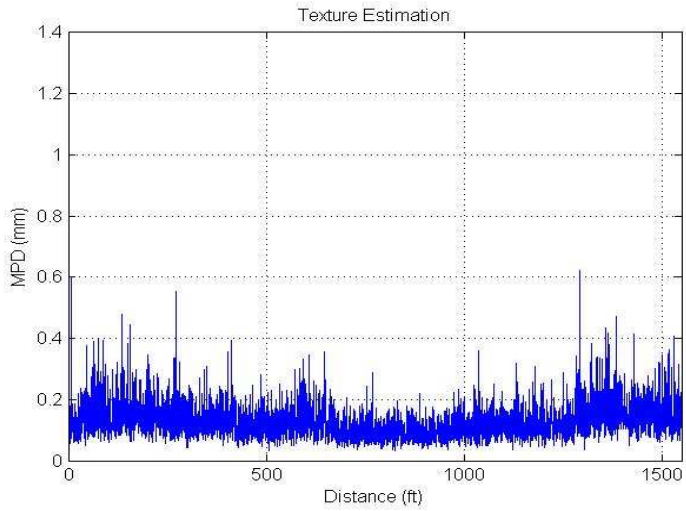


Figure 6.9 MPD Values for Estimating Texture Contents of the Tested Section

6.2 Design, Implementation, and Evaluation of Real-time Road Surface Analyzer system

6.2.1. System Modeling using UML and UML Profiles

UML is a graphical language used to specify, visualize, construct and document object-oriented systems by providing a set of diagrams that help in presenting such systems. Some of the common UML diagrams are class diagrams. These are static structure diagrams which describes the system's structure by showing the classes, their attributes, and the relationships between them. Another diagram is the state-chart diagram that represents system's behavior by showing a state machine with all possible states for an object to be in. The use case diagrams are graphical overview of the functionality provided by a system in terms of actors, their use cases, and any dependencies between those use cases. Also, sequence diagrams are kind of interaction diagram in UML, which shows how processes operate with each other and in what order. In the following subsections use case, class, and state-chart diagrams representing the system to be modeled will be discussed.

6.2.2. Use case representation diagram

As part of system analysis, the first step of modeling any system using UML is to create a use case diagram. Since the use case diagram deals with the system to be modeled as a black box it describes the external properties of some features in the system without dealing with any internal implantation details. Figure 6.10 is the use case diagram for the road surface analyzer. The diagram shows the relationship between the actors and the use case. The actors here are the road surface itself and the staff that is responsible for dealing with the data set and the results to schedule maintenance operations if the road requires one. Also, the use case diagram demonstrates the relationship among the use cases which are the operations to be performed.

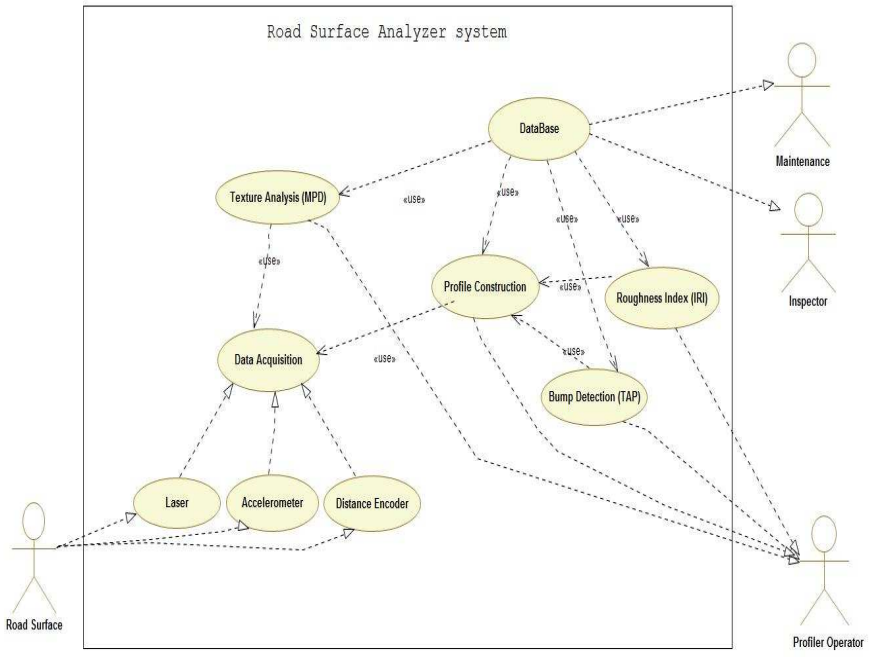


Figure 6.10 Road Surface Analyzer Use Case

6.2.3. Processor Class representation

Figure 6.11 shows the general class (super-class) core processor which represents a generalized modeling of a processor object. Four subclasses were created to for each core of

the quad-core embedded system. The core objects will be used as part of the embedded device class discussed later.

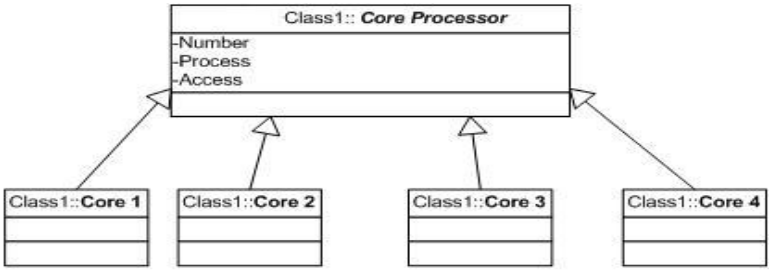


Figure 6.11 Core processor super-class generalization

The HRM diagram in Figure 6.12 is created to represent a quad core processor (either Core 2 Quad or the Quad core Xeon) with all of its hardware components from individual core, L2 cache, and bus interface with specifications that can be changed according to the processors' model from either of the quad core processors' family.

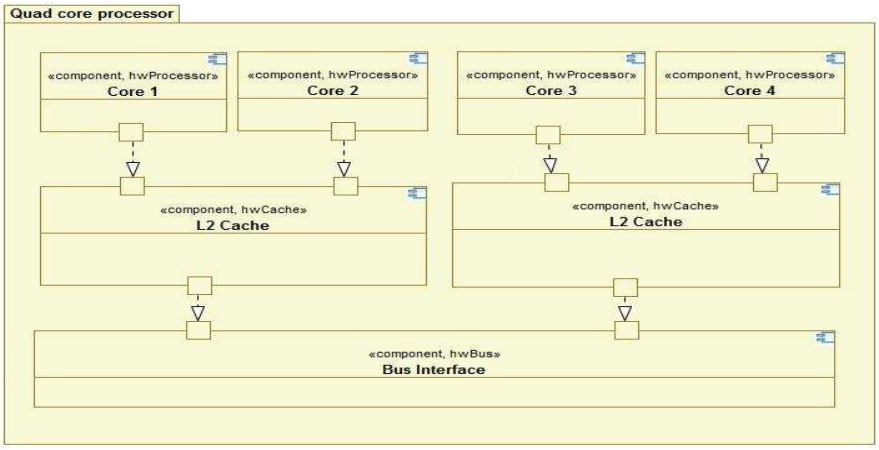


Figure 6.12 HRM Diagram for Intel Quad Core Processor Modeling

6.2.4. Parallel processing modeling

Mapping the tasks carried by the system here to a multi-core processor means that the tasks are processed in parallel. UML models parallelism in two ways. First, all objects are considered to be parallel entities. Second, a single object entity exhibits a concurrent behavior. This means that the object's state-chart is specified as a set of concurrent components. Figure

6.13 shows the state-chart for the core 1 object. Core 1 is supposed to perform two operations; the first one is to collect the sensor readings, while the second operation is to perform texture analysis once enough data is obtained (100 mm worth of data). Those two operations can be performed concurrently.

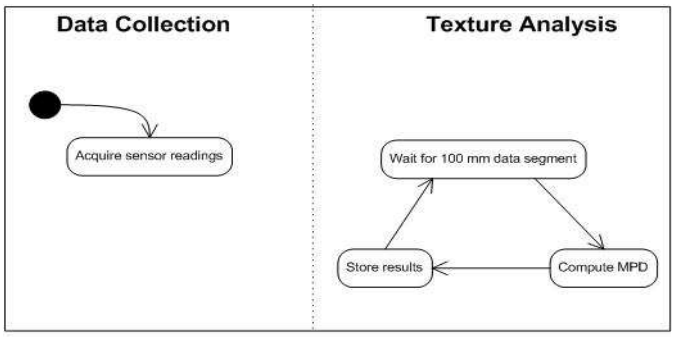


Figure 6.13 State-chart of first core tasks

6.2.5. System Modeling

The embedded device class in Figure 6.14 contains four objects of type core processor class representing the quad core system used. The diagram also shows the relationship between the main processing unit and the way data is acquired and stored. The data retrieved by the embedded device class from the sensor class through the analog-to-digital converter class is then mapped to the appropriate core objects. In the final system implementation processor affinity will be used to guarantee that the tasks are executed by the core processor assigned to perform the task during the analysis and modeling phase.

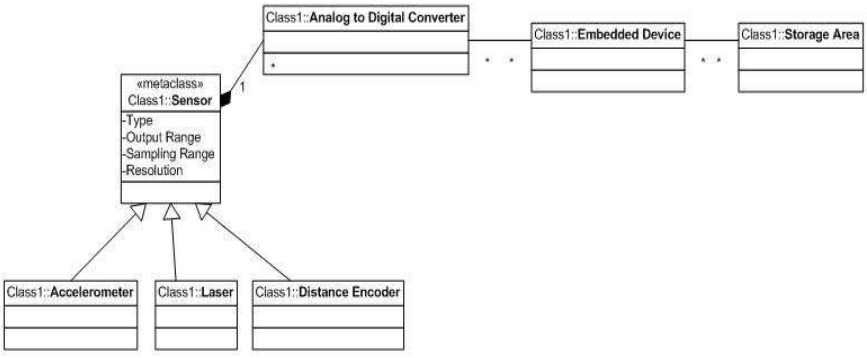


Figure 6.14 Road surface analyzer class diagram

Figure 6.15 illustrates the HRM model of the road surface analyzer system. Using the stereotypes defined by MARTE for different hardware the processing unit (CPU) represented here is one in which any processor can be used for such system representation mainly any quad core processor such as the one modeled in Figure 6.12. This provides an aid to model and specify the hardware devices and the controllers important to build such as system. On the other hand, the software functions and methods to be used are modeled with the aid of SRM profile as in Figure 6.16.

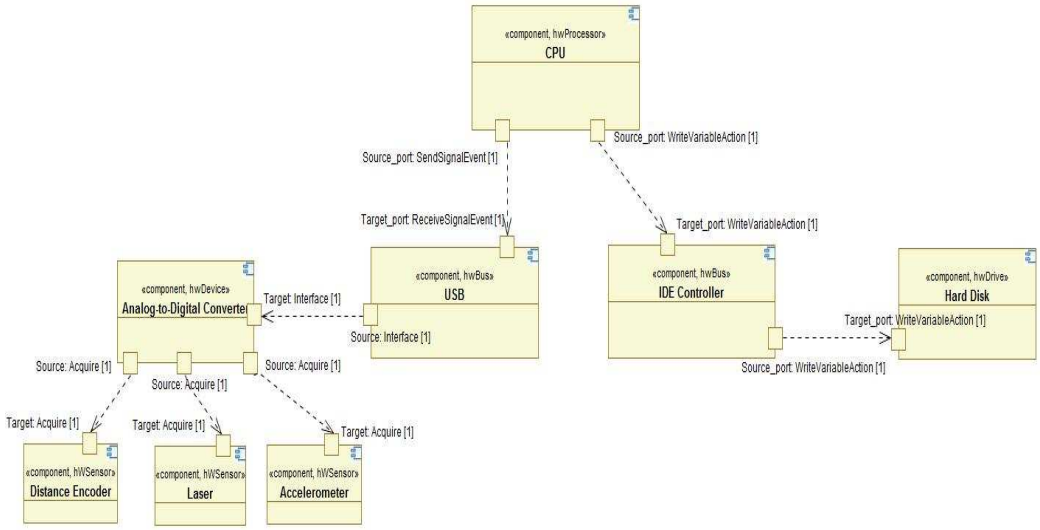


Figure 6.15 HRM Diagram for Road Surface Analyzer System

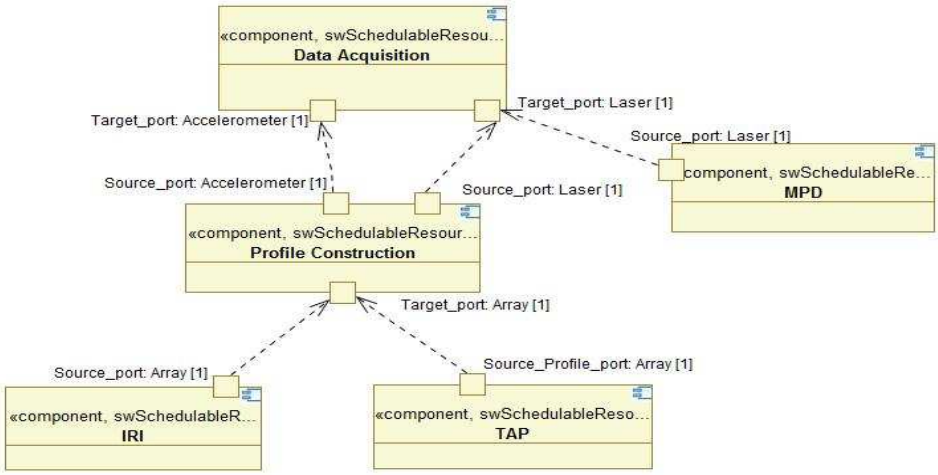


Figure 6.16 SRM Diagram Representing the Road Surface Analyzer System

6.2.6. State transition Diagram

In this subsection the state transition (or state-chart) diagram of the system is introduced as in Figure 6.17. The figure show how processed data are distributed among different processes (or threads) using the fork pseudo-states. The fork pseudo-state is a connector that branches a single input transition into multiple outgoing transitions to different states that will be activated concurrently. While the join pseudo-state joins together multiple incoming transitions into a single transition.

In terms of software implementation, the application is programmed in C++ with the aid of OpenMP as a threading technology. The threads created according to thread mapping obtained from decomposing the given application state diagram as shown in Figure 6.18 the tasks to be processed are mapping according to dependency and amount of work to be done in each stage; the mapping or the problem decomposition fits the data flow decomposition and more precisely, the producer/ consumer model. As shown in the figure, there five tasks, the main task is data acquisition. The second task is the texture analysis, third one is profile construction. Both second and third task depends on the first task and the data passed to them from the first task, in which the relationship among them fits the producer/ consumer model where the data acquisition represents the producer and both the texture analysis and profile

construction are the consumer part in this relationship. The results from the profile construction task will be passed to two other tasks and they are the bump detection and the IRI computation; in which these three tasks also follow the producer/ consumer model. This way of decomposition requires creating five different threads each of them is assigned to one of the tasks. And according to performance analysis done in chapter 3, core i7 processor is most appropriate to be used.

In another way of decomposition first and second tasks (data acquisition and texture estimation) can be merged to form a single task without changing the other three tasks. In this way, only four threads are required.

Figure 6.19 lists the main part of the code that starts with setting the number of threads to be used (5 threads) then the program will call the ADC() function to initialize the data acquisition unit and any other device that needs initialization) . After that the program will create the four threads with the aid of section pragmas in OpenMP were each one of them will execute one task assigned by the function calls in each section.

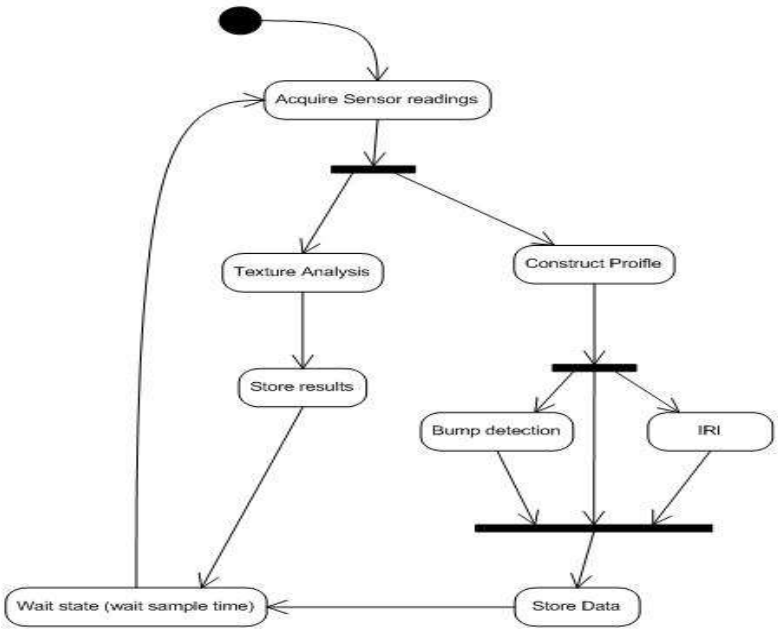


Figure 6.17 State-chart diagram

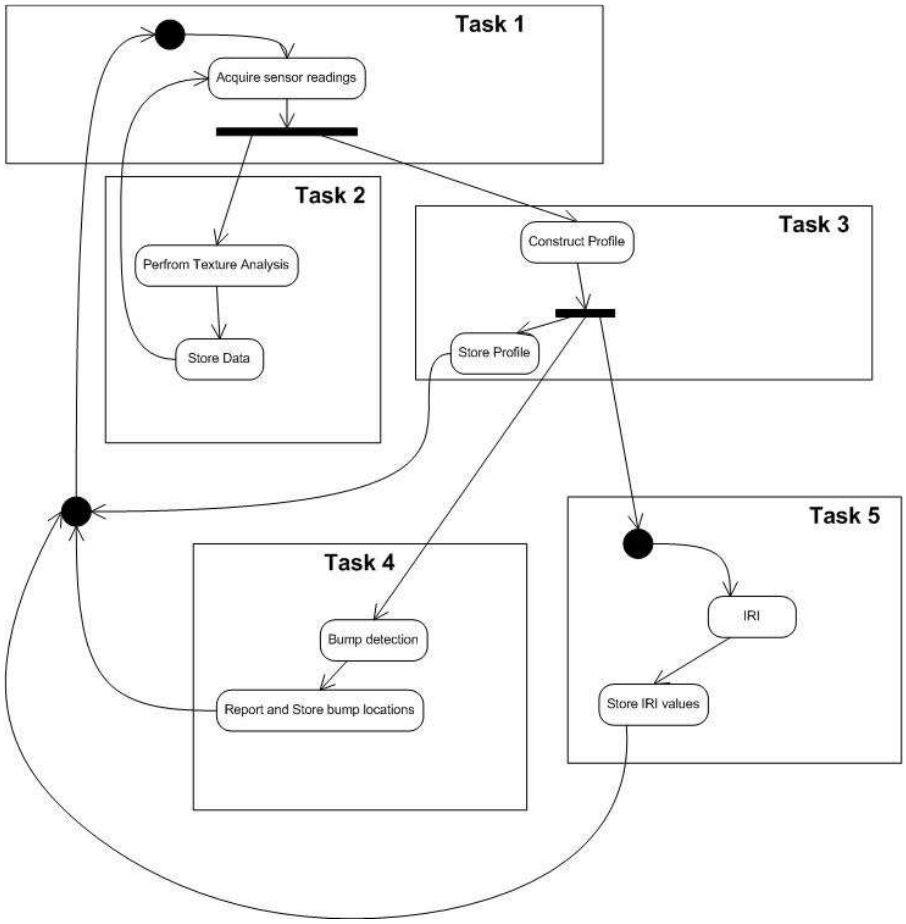


Figure 6.18 Road surface analyzer system state diagram (Thread mapping shown)

```
#include "stdio.h"
#include "math.h"
#include <time.h>
#include <omp.h>
...
void Set_ADC(); //Initialize the ADC Device
void Data_Acquisition();
void Construct_Profile();
void Texture_Analysis();
void Bump_Detect();
void IRI();

int main()
{
omp_set_num_threads(5);
Set_ADC();
#pragma omp parallel sections
{
 #pragma omp section
          Data_Acquisition();
#pragma omp section
          Texture_Analysis();
 #pragma omp section
     Construct_Profile();
 #pragma omp section
     Bump_Detect();
 #pragma omp section
          IRI();
          }
 printf("Program End...\n");
}
```

Figure 6.19 Sample Code for Implementing the Road Surface Analyzer in C++ with OpenMP

6.3 Design and Implementation of Real-time three Dimensional (3D) Road Profiler System

The system described in this subsection is designed in order to collect data required to generate a 3-D image of any road surface.

The hardware implementation of this system consists of three parts the motor controller system which is responsible of movements and locating the laser sensor to the desired location. It consists of the Compax3 system with the linear actuator (BLMA 120.) The motor part which is mainly a bar of 14- foot length is shown in Figure 6.20. The measurement bar is installed in front of truck. The second part is responsible for collecting the data from the sensors. It consists of

the sensors (laser and gyroscope shown in Figure 6.20), Data translation card and the laptop or embedded PC.

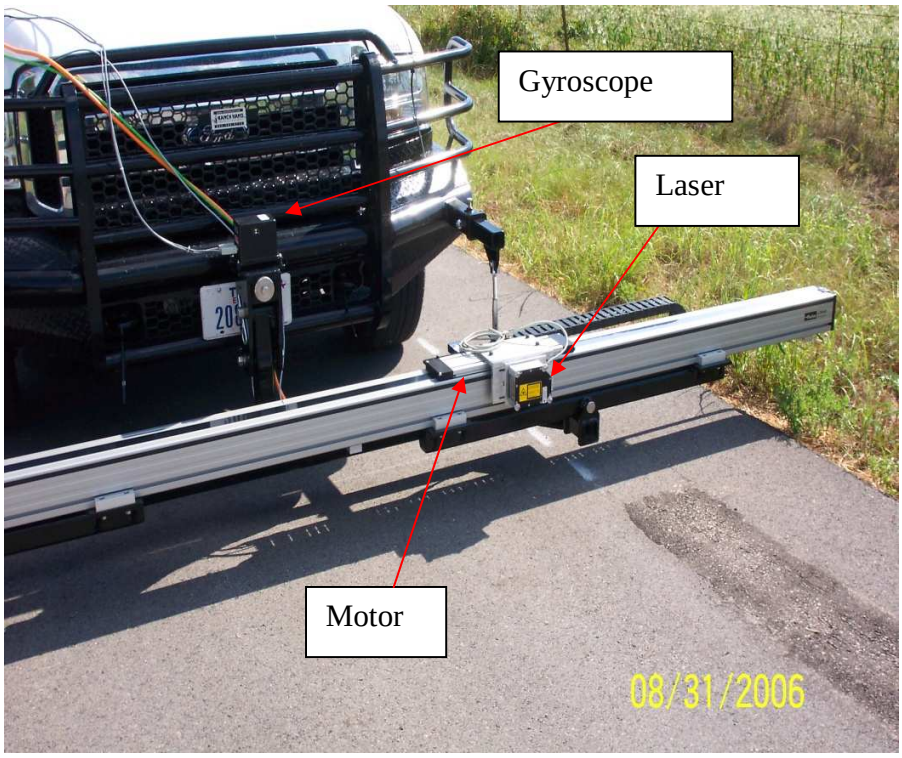


Figure 6.20 Laser and Gyroscope sensors

A static 3D profiler is, when the vehicle is not moving can be implemented using linear actuator, one laser sensor and a gyroscope. The system is similar to the one shown in Figure 6.20. The system was used to scan sections with different rut depths (Figures 6.21 and 6.22.) For this system, the testing vehicle will be parked at a preset location in the section to be scanned for data collection. The liner actuator is used to provide fixed step movements across the road surface and at each step laser reading is taken representing the distance between the vehicle and the road surface. Once a cross section of the road is scanned the vehicle is forwarded for 9.5 inches to scan the next strip. This procedure is repeated 27 times in each section. Check [65] for more information about the system and its implementation.

Figure 6.21 Test Section with rut depth from 0 - 1.0 inch

Figure 6.22 Test Section with rut depth from 0.5- 1.5 inch

The data collected is then filtered and a plot is generated to represent the section in 3 D image. A 3×3 and 5×5 Median filters were used to construct a smoother representation of the road surface. Figures 6.23 and 6.24 show the construct 3D image of the road surface as well as the filtered images.

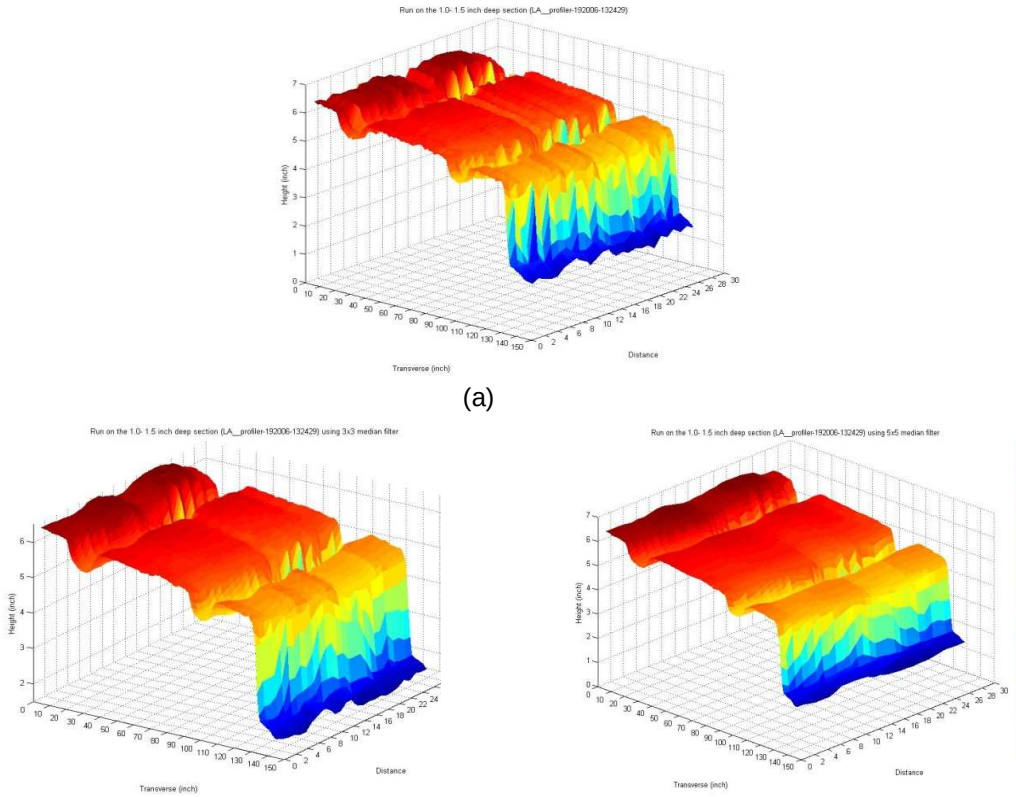


Figure 6.23 3-D images constructed from laser data collected for the road surface (a) Original Dataset for 0.5- 1.5 inch section, (b) Result of Using 3×3 Median Filter, and (c) Result of Using 5×5 Median Filter

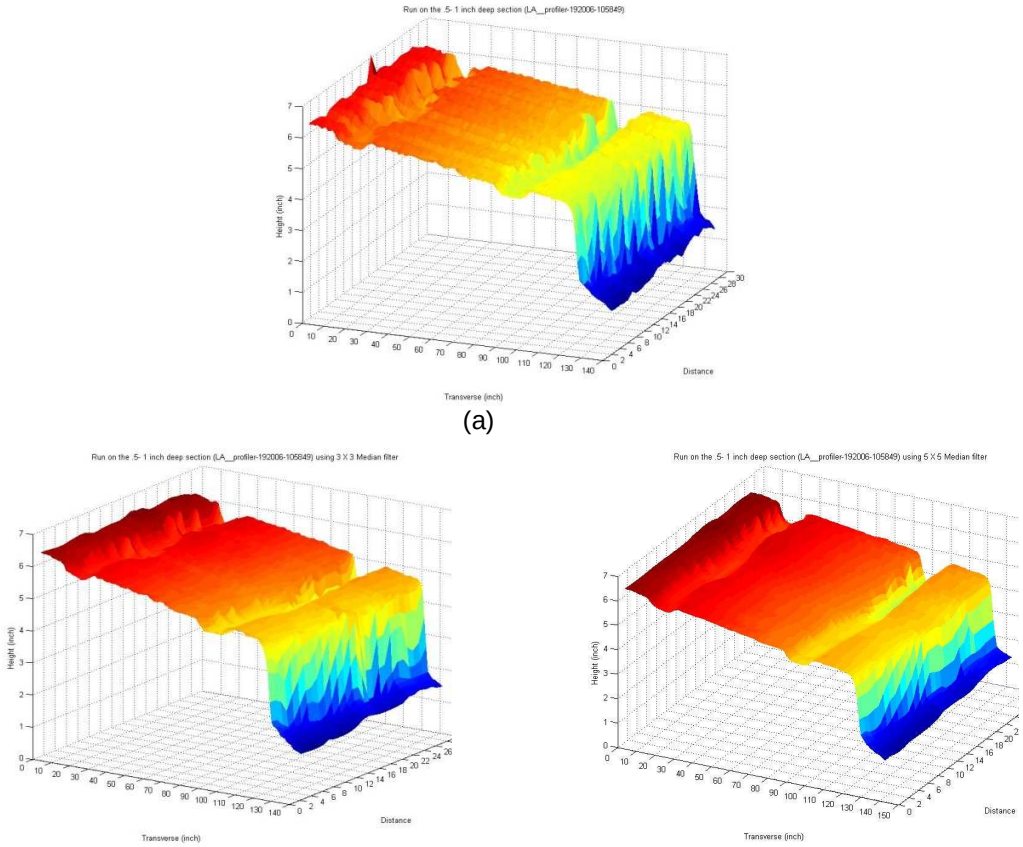


Figure 6.24 3-D images constructed from laser data collected for the road surface (a) Original Dataset for 0- 1 inch section, (b) Result of Using 3×3 Median Filter, and (c) Result of Using 5×5 Median Filter

CHAPTER 7

CONCLUSION AND FUTURE WORK

This research effort addresses many of the complexities involved in developing a real-time multi-core embedded system. Recently, increased performance in computing components is being achieved through the use of multi-core processors. In order to make use of the full power of these devices, multi-threading and parallel computing concepts are required. Parallel processing, once only prevalent in large servers and other intensive computing machines, are now beginning to appear not only in most servers, desktop and notebook computers, but also in embedded systems. Although opening up a host of new capabilities and applications for the small embedded processor, proper use of these devices present many challenges to the embedded designer. This research addresses many of these challenges, using a road profiling system as a case study.

This research investigates and establishes the steps to design, analyze, and implement a real-time embedded multi-core application, a real-time road surface analyzer (road profiler). The road profiler is an instrument used by transportation engineers for accurate measurement of road or pavement profile. From this profile various statistical variables can be computed, providing the condition or serviceability of the pavement. The road profiler is analyzed, redesigned, and implemented from a computer engineering perspective and as an embedded application, adding concurrent processing of several of the statistical pavement characteristics used for specifying the pavement conditions.

This research analyzed the requirements of the system; defined and designed the appropriate tools needed for this system, and then implemented the system in order to perform its tasks in real-time with the aid of a multi-core embedded system.

Software engineering tools were used to model the system which is done with the aid of UML modeling language and UML extensions or profiles. The UML and UML profiles were first explored and a number of common UML and/ or UML profiles modeling practices were concluded in terms of the appropriate usage of such modeling tools:

- Use of UML profiles only when standard UML cannot perform the task.
- Use SysML diagrams for general modeling of the system.
- Use MARTE for modeling the system's details.
- Use MARTE specifically to model the hardware system with all of its details along with modeling the software methods which are part of the application.

Selecting the correct embedded processor for any embedded application under development requires evaluating the performance benefits of the processor. In this research, several embedded multi-core processors were evaluated in order to select which one can provide the optimal performance for this application. The processors' performances were evaluated using some of the well known benchmarks, which consist of several tasks to test key processing factors from memory usage, execution time, and data transfer when using different number of threads.

The evaluated processors vary according to the speed; power consumption, cache levels and sizes, and also the implementation of parallelism in each one of them were discussed. Then their performances were tested and analyzed using some of the common benchmark suits in terms of memory access and bandwidth, throughput, and execution time. From the results it can be concluded that integrating the memory controller in the processors chip can increase the memory throughput as in the core i7 case. Also, it shows that integrating and implementing the multi-core (CMP) technology boosts the performance of the processor more than the usage of the hyper-threading technology (SMT). The Hyper-threading technology proves that although it can improve the performance of a given processor, care should be taken when using (enabling) this technology since for certain applications, not only

does it not have any effect on increasing the performance but also it may lead to a drop in the performance.

The usage of multi-core processors in embedded systems forced embedded application developers not to rely on the hardware alone to provide an increase in performance; developers must also select the appropriate software technologies that can be integrated effectively with the multi-core processors to take full advantage of the designed embedded system.

For this reason, embedded operating systems were investigated to select the appropriate one to be used in this application which is capable of meeting the real-time constraints and has sufficient capabilities to provide support to multi-core processors and parallel computation. Windows embedded standard is the main candidate to be installed in the embedded system as the real-time operating system, along with implementing the embedded application using C (or C++) programming language and OpenMP API to provide support for multi-threading.

This research also tested different types of filters to be used for profile computation and reached the decision that either a 3^{rd} or 4^{th} order high-pass IIR Butterworth filter can provide an accurate profiling computation taking into consideration that zero-phase filtering (reverse filtering) is needed to attenuate any nonlinear phase shift effect, mainly due to some road deficiencies, bumps or potholes and which can affect the profile computation for segments.

7.1 Future Work

- Use of different filter types for profile computation
- Test Line laser sensors and compare with the current single spot laser sensors as a solution of texture problems.
- Evaluate different embedded operating systems
- Build a benchmark to evaluate any candidate embedded board. The benchmark will simulate a profiler system in terms of collecting data and process it using different

computational algorithms to obtain the profile, IRI value, texture estimates, etc. Hardware in the loop (HAL) concept can be used to test and evaluate new embedded boards.

- Evaluate and compare different types of threading techniques
- Windows Threading APIs
- OpenMP
- Intel Threading Building Blocks (TBB)
- Modify 3D profiler to include multiple laser sensors, and check the feasibility of using line laser sensors such as the RoLine 1130/ 1140.

REFERENCES

[1] B. Selic, J. Rumbaugh, *"Using UML for modeling complex real-time systems"*, ObjecTime, March 1998

[2] A. Staines, *"A Comparison of Software Analysis and Design Methods for Real Time Systems"*, PROCEEDINGS OF WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY VOLUME 21 MAY 2007

[3] SysML, http://www.omgsysml.org/

[4] MARTE, http://www.omgmarte.org/

[5] http://www.omg.org/technology/documents/formal/schedulability.htm

[6] Y. Vanderperren, and W. Dehaene, *"From UML/SysML to Matlab/Simulink: Current State and Future Perspectives,"* Proc. Design, Automation and Test in Europe (DATE) Conf., Munich, Germany, 6-10 March 2006.

[7] Y. Vanderperren, and W. Dehaene, "UML 2 and SysML: an Approach to Deal with Complexity in SoC/NoC Design," Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05), Munich, Germany, 7-11 March, 2005.

[8] M. Mura, A. Panda, M. Prevostini, "Executable Models and Verification from MARTE and SysML: a comparative study of code generation capabilities", DATE'08, Workshop on Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile Proc. pp 29-34 Munich, Germany, 14 March 2007.

[9] M.Emilia Cambronero Gregorio D´ıaz J.Jos´e Pardo Valent´ın Valero Fernando L. Pelayo, "RT-UML for modeling Real-Time Web Services", Proceedings of the IEEE Services Computing Workshops (SCW'06)

[10] P. Boulet, P. Marquet, E. Piel, and J. Taillard, "Repetitive Allocation Modeling with MARTE," in Forum on specification and design languages (FDL'07)

[11] http://en.wikipedia.org/wiki/Class_diagram

[12] Intel Core 2 Quad, http://www.intel.com/products/processor/core2quad/

[13] Intel Core i7, http://www.intel.com/products/processor/corei7/index.htm

[14] L. Peng, J-K. Peir, T. K. Prakash, C. Staelin, Y-K. Chen, D. Koppelman, "*Memory Hierarchy Performance Measurement of Commercial Dual-Core Desktop Processors*", In *Journal of Systems Architecture*, vol 54(8), Aug. 2008, page 816-828.

[15] http://en.wikipedia.org/wiki/NAS_benchmarks

[16] http://www.cs.virginia.edu/stream/

[17] L. McVoy, C. Staelin, *"lmbench: Portable tools for performance analysis"*, 1996 USENIX Annual Technical Conference, pp 279- 294

[18] Ryan E. Grant and Ahmad Afsahi, "*A Comprehensive Analysis of Multithreaded OpenMP Applications on Dual-Core Intel Xeon SMPs*", Workshop on Multithreaded Architectures and Applications (MTAAP'07), In Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007), Long Beach, California, USA, March 26-30, 2007.

[19] http://edc.intel.com/Platforms/EP80579/#platform-overview-content=platform-overview-toggle~~visible-content

[20] Intel Tolapai, http://www.intel.com/design/intarch/ep80579/index.htm

[21] http://www.intel.com/design/intarch/devkits/5100/index.htm

[22] Baily, D.; Barscz, E.; Barton, J.; Browning, D.; Carter, R.; Dagum, L.; Fatoohi, R.; Fineberg, S. *et al*., "The NAS Parallel Benchmarks", *NAS Technical Report RNR-94-007*, March 1994, NASA Ames Research Center, Moffett Field, CA

[23] http://en.wikipedia.org/wiki/NAS_benchmarks

[24] Frumkin, M.; Shabanov, L. "Benchmarking Memory Performance with the Data Cube Operator", *NAS Technical Report NAS-04-013*, September 2004, NASA Ames Research Center, Moffett Field, CA

[25] http://evanjones.ca/smt-performance.html

[26] http://ark.intel.com/Compare.aspx?ids=34311,34695,29765,37147,35635,35641,

[27] H. Jin, M. Frumkin, and J. Yan, "*The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance*", *NAS Technical Report NAS-99-011*, NASA Ames Research Center, Moffett Field, CA, Oct 1999.

[28] http://www.scientificcomputing.com/parallel-processing-speeds-visualization.aspx

[29] Differences Between Windows XP Embedded and Windows XP Professional

[30] http://www.microsoft.com/windowsembedded/en-us/products/westandard/faq.mspx

[31] http://zone.ni.com/devzone/cda/epd/p/id/5956

[32] http://www.dpie.com/software/winxpembedded.html

[33] Multi-Core Programming Increasing Performance through Software Multi-threading, Shameem Akhter, Jason Roberts, Intel Press, 2006

[34] http://www.mathworks.com/products/xpctarget/?BB=1

[35] http://www.mathworks.com/products/xpcembedded/

[36] http://software.intel.com/en-us/blogs/2006/10/19/why-windows-threads-are-better-than-posix-threads/

[37] http://en.wikipedia.org/wiki/Windows_CE_5.0

[38] http://www.lynuxworks.com/

[39] http://www.qnx.com/

[40] http://www.rtlinuxfree.com/

[41] http://www.streambag.se/files/rtproj.pdf

[42] http://en.wikipedia.org/wiki/Windows_Embedded_Standard_7

[43] http://en.wikipedia.org/wiki/RTAI

[44] http://en.wikipedia.org/wiki/RTAI

[45] http://www.diamondsystems.com/products/athenaii

[46] Software Development for Embedded Multi-core Systems: A Practical Guide Using Embedded Intel Architecture, Max Domeika, Newnes, 2008

[47] http://standards.ieee.org/regauth/posix/index.html

[48] http://www.threadingbuildingblocks.org/

[49] Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism, James Reinders, O'Reilly Media; 1 edition, 2007

[50] http://software.intel.com/en-us/intel-vtune/

[51] http://software.intel.com/en-us/intel-thread-checker/

[52] http://software.intel.com/en-us/articles/intel-threading-building-blocks-openmp-or-native-threads/

[53] R.S. Walker, E.G. Fernando. *Pilot Implementation of Bump Detection Profiler*. Research Report. 5-4385-01-1. University of Texas, Transportation Instrumentation Laboratory, Arlington, TX. December 2007.

[54] R. Walker, E. Fernando, and Y. Sho, "*Develop a Methodology For Establishing Bump Detection Using Inertial Profile Measurements For Implementation With The Ride Specification*", Texas Department of Transportation (TxDOT) Research Report 4479, April 2004 .

[55] M. W. Sayers and S. M. Karamihas, *the Little Book of Profiling,* University of Michigan Transportation Research Institute, September 1998.

[56] E. B. Spangler, W. J. Kelly, "*GMR Road Profilometer, a Method for Measuring Road Profile,*" Research Publication GMR-452, General Motors Corp., Warren, MI, Dec 1964

[57] ASTM E1845-01,"Standard practice for Calculating Pavement Macrotexture Mean Profile Depth", 2005

[58] Dyer S. A. and Dyer J. S., "Implementation Problems in Inertial Road-Profiling: An Overview", IEEE International Instrumentation and Measurement Technology Conference I^2MTC 2008, Victoria, Vancouver Island, Canada, May 12- 15, 2008.

[59] http://www.roadprofile.com/

[60] Jareer Abdel-Qader, Emmanuel Fernando, and Roger Walker, “Current Investigations into the Effects of Texture on IRI”, RPUG 2008, Austin, TX

[61] Roger S. Walker, Emmanuel Fernando, and Cindy Estakhri, “Flexible Base Ride Specification Development and Evaluation”, TxDOT 0-4760, 2008

[62] http://www.mathworks.com/access/helpdesk/help/toolbox/signal/f1-859.html

[63] R.S. Walker, E.G. Fernando, “*A Portable Profiler for Pavement Profile Measurements - Interim Report*,” Technical Report. 0-6004-1. Texas Transportation Institute, College Station, TX. May 2009.

[64] http://www.diamondsystems.com/files/binaries/AthenaII-Datasheet.pdf

[65] Walker, Roger S., Emmanuel Fernando, Eric Becker, Jareer Abdel Qader, Gerry Harrison, "*Using Profile Measurements to Locate and Measure Grind and Fill Areas to Improve Pavement Ride",* Texas Department of Transportation, Preliminary Publication, March 2007.

BIOGRAPHICAL INFORMATION

Jareer Abdel Qader was born in Kuwait, in 1971. He received his B.Sc. degree from AL-Ahliyya Amman University, Jordan, in 1996. His M.Sc. degree from Jordan University of Science and Technology, Jordan, in 2000, and Ph.D. degree from The University of Texas at Arlington in 2010, all in Computer Engineering. Jareer worked as computer engineer at AL-Ahliyya Amman University, Jordan during the period from 1996 to 2000. After that he joined the department of computer engineering, at Jordan University of Science and Technology, Jordan, as a full-time lecturer until the end of 2003.