EMBEDDED LINUX FOR HEALTHCARE APPLICATIONS

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

MASTER OF TECHNOLOGY
IN
VLSI DESIGN & EMBEDDED SYSTEMS

Submitted by:

Ishita Sagar

2K19/VLS/06

Under the supervision of

Dr. Malti Bansal
Assistant Professor

ELECTRONICS & COMMUNICATION ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

2019-2021

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Ishita Sagar, Roll No. 2K19/VLS/06, student of M.Tech (VLSI & Embedded systems), hereby declare that the project Dissertation titled "**EMBEDDED LINUX FOR HEALTHCARE APPLICATIONS**" which is submitted by me to the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi **(Ishita Sagar)**

Date:14/06/2021

ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled **"Embedded Linux for Health Care Applications"** submitted by Ishita Sagar, 2k18/VLS/06, Electronics and Communication Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is a record ofthe project work carried out by the student under my supervision. To the best of myknowledge this work has not been submitted in part or full for any Degree or Diplomato this University or elsewhere.

Place: Delhi

Date**:** 14/06/2021

Dr. Malti Bansal

Assistant Professor

ACKNOWLEDGEMENT

A successful project can never be prepared by the efforts of the person to whom the Project is assigned, but it also demands the help and guardianship of people who helped incompletion of the project. I would like to thank all those people who have helped me in this research and inspired me during my study.

With profound sense of gratitude, I thank Dr. Malti Bansal, my Research Supervisor, for his encouragement, support, patience, and her guidance in this project work. Furthermore, I would like to thank my mentors at my internship who helped me throughout to learn and understand new concepts and work. I would also like to thank my friends and family for their constant support.

I take immense delight in extending my acknowledgement to my family and friends who have helped me throughout this project work.

Date:14/06/2021 **Ishita Sagar**

CONTENTS

List of Figures

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In today's fast pace world, healthcare has become the most important and care-worthy aspect of human life. Everyone wishes to stay fit and healthy and get the correct and fast-paced treatment if diagnosed with any disease. Without the provision and access to proper healthcare, the human life is at risk.

Healthcare in today's date has become very advanced and effective. This advancement in achieving great healthcare services and provision, is owed to the interference of technology in the healthcare areas. The healthcare future has shaped and continues to shape in front of our eyes as technology interferes with it. The future and excellence of healthcare moves hand-in-hand with technology. Many different areas of science and engineering are involved in multiple areas of healthcare. One such area that has helped develop medical equipment to a great extent is Embedded Linux.

An Embedded System refers to a computer hardware and software combination which is designed to perform any specific functionality. Embedded systems can also function under the hood of larger systems. As embedded systems are computing systems, they range from having minimal or no user interface, to having a highly complex one. Now, moving further, it is possible to combine Embedded Systems with the Linux operating systems to achieve great heights. Embedded Linux refers to a category of Linux operating system or kernel which is created to be installed and operated inside of embedded devices and other appliances. It is nothing but a cut-down or trimmed version of Linux, usually for low power consumption. We can call it as a compact and smaller version of Linux operating system that provides all the necessary features according to the embedded system requirement.

Embedded Linux has made its way through the healthcare domain. There are many reasons for the vast use of Embedded Linux in healthcare applications. Firstly, Linux is an open-system operating system which can offer multiple applications over numerous

platforms. However, there are specific requirements and restrictions which are imposed for Linux when it is used in medical systems and equipments.

1.1 Linux Overview

Linux operating system was launched in 1991 as an alternate option to the Unix operating system. 'Linux' as known today was developed using the kernel from Linus Torvalds in combination with the free software by GNU project, which was started by Richard Stallman. When the linux kernel was combined with GNU system, it gained immense popularity majorly due to its open-source nature and availability. Moving forward, the system produced numerous versions and distributions, from vendors including Fedora, Ubuntu, Red Hat etc. and even completely open-source distros including Linux Mint, Debian, Slackware etc.

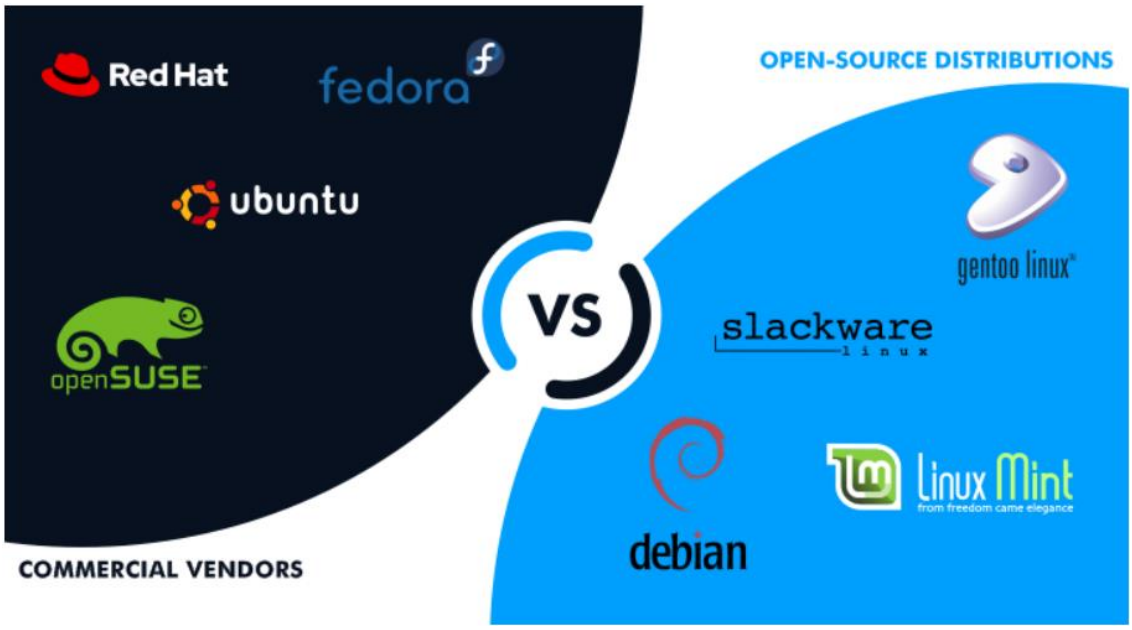

Figure 1. Linux commercial vendors vs open-source distributions

Linux is compatible with several varieties of hardware platforms, due to its portability. It needs moderate demands, such as 8 or 16 Mb of RAM, 32 Mb of flash, and a small ARM core. These are the minimum requirements for Linux hardware, and they increase depending upon the kind of application. In order to run any custom ized hardware, any

operating system requires a customized firmware image. The same rule is also applicable for Linux. While building a custom Linux image, one can face issues in kernel configuration or with the availability of drivers. Linux allows for several modifications with the choice of multiple applications in combination with security patches. The global Linux experts ensure the security of the operating system, making it resistant to any kind of malware or virus.

1.2 Linux for Medical Devices

The medical or the healthcare industry has adopted the integration of high tech solutions for its products and procedures. However, the rise of this adoption practice has slowed down and saturated due to the medical industry being highly and regularly regulated, and the scope of any update or change with ancient methods becomes a challenge for any medical device company. The manufacturing company needs to ensure that the product completely follows and meets the required relevant standards and the regulatory documents as well. When it comes to a medical device, it also needs to follow the requirements of safety and security performance. This entire process is a tedious one and not a very budget friendly one. This is the major reason for many manufacturers of medical devices to have been keep using the old software and hardware for decades.

The operating system of Linux is a sound alternative for both porting of a system from the older versions of software platforms and also the complete and effective development of a new product from the scratch. This is one of the reasons that Linux operating system has been chosen by many companies.

The entry of Linux in the medical industry is owed to the open-source nature of the operating system. It helps in cutting down costs on the development and maintenance of any medical device or software. Linux can combine both general purpose operating system and real time operating system, thereby covering the entire range of medical applications ranging from a simple one to a life-critical application or use. Linux effectively provides high levels of security, that upholds the tight standards of safety-

sensitive aspects of the medical industry. It is a perfect fit for smooth working of equipment that support life or even transmission of any patient's data with utmost security. Finally, Linux copes well with healthcare regulations. The software can either be developed separately in the form of a medical device, or it can also be embedded or fit inside of a medical equipment or device.

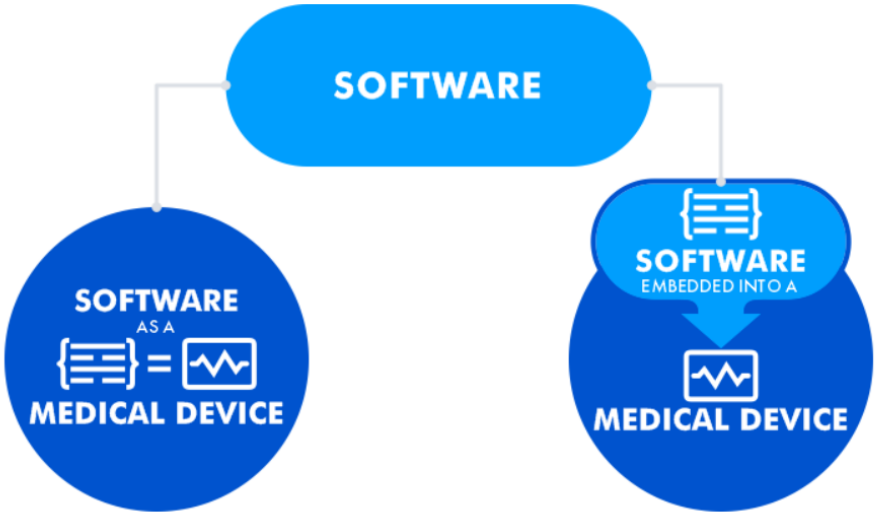

Figure 2. Software Development for medical purposes

1.3 Linux features helpful in Medical Devices

It is apparent that an open-source answer is tons greater appealing from the industrial factor of view than a proprietary running system. It is obvious that one of these challenge gives advantages on implementation and maintenance, however it also helps in saving costs for the license and support. Linux offers immense customization due to the flexibility of the operating system. This open-source platform provides a huge variety of pre-present designs that can fulfill the specific requirements of the software or design. Necessary adjustments can be made to the code whenever needed. Features can be easily added or removed in order to make the design more functional, clean and effective. Special development platforms are available which are helpful in creating

customized images for any embedded system, irrespective of the target hardware. Yocto-project is a common example for such a platform. Yocto-project provides a very wide-range of open-source tools with proper support for developing embedded linux which can successfully run over numerous architectures including ARM, MIPS, Intel etc. Security is one of the major advantages offered by Linux. Linux being malware resistant, stands against cyber-attacks, hence protecting the personal data of any patient. Any issue related to security can be easily and quickly identified and resolved. Linux systems are armed with encryption technologies. Since linux can support both general purpose operating system and real time operating system, it makes it an ever more suitable option. General purpose option for linux is a good option for a simple portable device which does not require large memory, real time performance and also limited time response. However, today's technology demands high accuracy and reliability from embedded medical devices, with additional low power consumption. Such devices are to be designed such that they can run on high speeds and also run any real time application. In such cases, real time capabilities can be used in embedded linux for medical devices. Such real time systems can include life support systems, which constantly monitor the patients' condition, and hence actions have to be performed at specific instants of time. Missing any such instant of time can be fatal for the health of the patient.

Owing to the high portability feature of Linux, when any system has to be moved over to a new hardware, the software can be ported easily. Linux is also compatible with outdated hardware. An operating system that is based on the new linux kernel is easy to run an older hardware. Linux being flexible, finds its application in major areas ranging from desktops, networks, servers, to embedded medical product and devices. It can be used in a small sample measuring device or even in complex medical devices such as MRI or a CT Scanner. Going with Linux can be considered as a forward-thinking plan or strategy. The product viability becomes very essential, when weighed against Windows platform end of life. We can avoid all such issues with Linux. Since linux is

globally developed and supported, linux has higher longevity. Linux zealots continuously develops newer version of the linux operating system, coming up with new tools, features, applications and the support which is needed in the long run. However, the open source nature of linux can also be a concern when it comes to certification.

1.4 Objective

This thesis talks about four medical devices, Neptune, Core2, Multigen and Sonopet which are based on Embedded Linux. A generic board present in all the four products, called the SOM board, fits well in these. The SOM board currently being used uses the processor i.MX53 by NXP, which is going to be obsolete soon. The current processor is being replaced by the i.MX 8M, another processor by NXP. Compatibility of the application of all the products needs to be checked. Basic testing such as benchmarking is also being carried out, using evaluation board, which is the MaaXBoard by AVNET. The MaaXBoard is a low-cost board, based on the NXP i.MX 8M processor. It is a single board computer, which stands to be ideal for applications such as Embedded Computing. The evaluation board is brought up , basic communication testing is done for the board. Benchmark using Sysbench is carried out, and once the benchmark results are obtained, the application software for one of the products, Neptune is ported to the evaluation board and tested if it gets built and run successfully.

1.5 Organization of the report

This report has been divided into 5 chapters. Chapter 1 introduces the overlap of medical devices and Embedded Linux. It also includes objective of the report. Chapter 2 talks about the Literature Survey. Chapter 3 talks about the practical work, the test results and application testing. And in Chapter 5 Conclusion and Future Scope is given.

CHAPTER 2

INTRODUCTION

2.1 Embedded Systems

The definition of an Embedded System can be stated as a combination of both computer hardware and software which reside inside a much bigger system or device. The device or systems completely encapsulates the software and hardware, usually making believe that we cannot recognize the embedded system inside the appliance. The embedding device is usually controlled by such systems. Several other tasks include communication between the multiple equipment pieces. The functionality of embedded systems is limited to few restricted options, usually they are created for performing a specific functionality, which is contrary to simple desktop computers.

When a general purpose computer or a desktop computer is the manufactured, it is not known what the customer will be using it for. They can use it for simple purposes such as playing games, as a server or anything else. However, in embedded systems, both the hardware and software are designed to deliver specific applications. Whereas a desktop computer consists of multiple embedded systems. When embedded systems are used for controlling purposes, they are bound under stringent timing conditions. In case important calculations are not completed within the defined deadline, the machine or even the part being worked upon can be damaged. Real time deadlines are specifically very important to meet.

Going with the definition, software and a processor are both present inside an embedded system. Other mandatory components are also present, like ROM or a flash chip for permanently storing the software, and RAM in order to store the run time data. According to the storage requirements of the developing system, both on chip or external memory devices can be used. Small devices have less than 1kB of RAM and ROM, whereas bigger systems have up to several megabytes of memory.

Apart from memory, embedded systems have input and outputs, to receive, process and send data. Figure 3 shows the setup of an Embedded System.

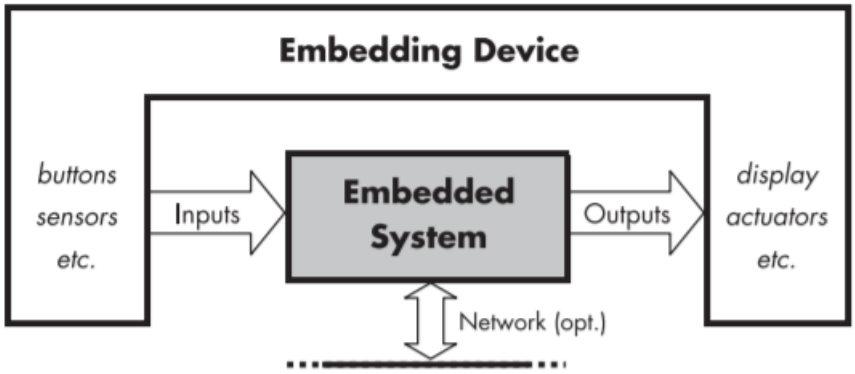

Figure 3. Embedded System Setup

Apart from the above mentioned features, which are common and mandatory in all embedded systems, the remaining embedded hardware is uniquely designed in order to achieve the specific requirements of the device. There are some embedded systems which have to be used with other such similar systems, and hence communication interface is needed between them. This can be achieved by some network type, like CAN, LIN etc.

Other design requirements include the following:

1. Processing Power: The register width needs to fit the requirement, in addition to the number of operation per second, MIPS. Processors from 4 bit width to 64 bit width are widely used in embedded systems. Like a small application of windows lift controller can operate using 4 bits, whereas an electronically controlled engine needs usually higher bits such as 16 or 32 bits or more.
2. Number of costs and Development Cost: These costs include the cost of the hardware as well as the software. It being a fixed cost, it becomes less important for higher volume products, whereas it becomes a dominating factor when the number of units to be produced are less.

3. Expected Lifetime and Reliability: A device is expected to function for several years, and also give effective performance and hence being reliable. Lifetime and reliability are important design decisions to be made, ranging from the selection of the components to the expenses for production and development.
4. Electromagnetic Compatibility (EMC): Owing to the close connections to the embedding device, there are standards and requirements which have to be met and considered for emissions and susceptibility of the electromagnetic disturbances, which have to be checked.
5. Environmental Requirements: The embedded systems also need to be resistant against many harmful influences in hostile environments. These influences can include dust, humidity, vibration, shock and temperature fluctuations. These factors have high importance in automotive applications.

2.2 CPU Architectures for the Embedded Systems

In general, the processor designed for an embedded system is chosen such as to meet and deliver the needs of any device. A processor's architecture has a big impact on the aforementioned requirements, especially in terms of computational power and cost.

2.2.1 x86

The x86 architecture family, sometimes known as desktop computer architecture, was introduced by Intel. However, a number of x86 processor are designed especially for the embedded systems. A “standard” x86-CPU requires an extra chipset, which typically consists of Southbridge and North-, to interface to peripheral devices or memory. There is just a lot of waste of resources, such as electricity and the space, when ordinary x86 CPUs are used in an average embedded system.

Embedded x86 processors, on the other hand, combine the chipset, CPU, and some extra glue code in a single integrated circuit. Figure 4 shows the architecture of x86 System-on-chip.

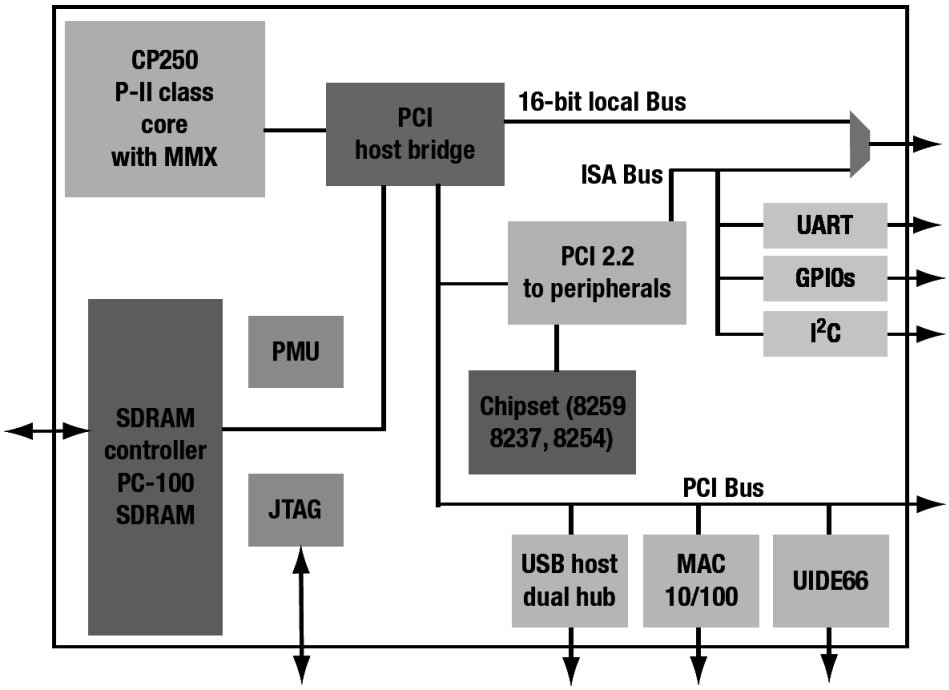

Figure 4. x86 System-on-chip

The x86 architecture is the one of the most well-documented amongst several architectures. The details of this architecture are covered in a number of books and online documents. When working on an Intel PC, development of the x86 is quite simple; cross platform development is not required (like most developers do). At work or at home, almost everyone has a computer with this architecture.

This architecture, however, only accounts for a minor portion of the typical embedded system marketplace. One factor is heavy cost of the x86-Processors, while other one is the architecture's comparatively more electricity consumption when compared with the another known architecture.

2.2.2 MIPS

The Stanford’s "Microprocessor without Interlocked Pipeline Stages" experiment gave

birth to the MIPS Platform. MIPS Technologies Inc. has licensed the CPU core as intellectual assets to numerous manufacturers.

Because this has so many different instruction sets, the OS and the software packages must be customized as per the target board.

Practical devices such as Sony's PlayStation 2 and 1, Nintendo's 64, SGI servers and workstations, as well as routers and other networking equipment, all use MIPS CPUs. According to MIPS technologies (source http://www.mips.com/), MIPS products can easily be found in 72 percent of VoIP (Voice over IP) applications, around 76 percent of cable set-tops, around 70 percent of the DVD recorders, and maximum cable modems.

2.2.3 Power PC

The PowerPC - architecture is the outcome through the teamwork between Motorola, Apple and IBM. Despite their origins in personal computing (Motorola's 68k CPUs being used by Apple Macs using) and also the mainframes (known Architecture of IBM called power), PowerPC processors were built for the use as quite high-performance processors and being embedded.

PowerPC is extensively based on the IBM's previous architecture called POWER, with which it has a high degree of compatibility. These 2 architectures have stayed close enough that the same programmes and OS will operate on both provided proper preparation is performed. The whole PowerPC instruction set is implemented in newer POWER series CPUs.

PowerPC CPUs are most frequently linked with Apple's MAC computers, but they also appear in a range of other devices, including the original TiVo hard-disk recorder and a number of set-top boxes. Like the x86, this also is very well documented and maintained by a wide range of OS.

2.2.4 ARM

The Advanced RISC Machine (ARM) - originally Acorn RISC Machine, like the MIPS architecture, is offered as intellectual property to chip makers. [30] ARM Ltd. are the one who solely creates CPU cores, which it then licences to companies like Atmel, Samsung and others. Marvell Technology Group also just purchased Intel's XScale series.

Unlike the MIPS's architecture, all the ARM's processors share the very same instruction set, ensuring that all of the processors are compatible to the software (assembly and binary code).

ARMv5TE and ARMv4 architectures are being used in almost all modern ARM CPUs. Only the clock speed, the cache, and some other characteristics differ.

ARM's processors are dominating in the mobile-electronics sector, where the low electricity consumption is a crucial design aim, due to their power-saving characteristics. According to ARM Limited, ARM has a market share of about 75% in the embedded RISC CPU market. ARM Partners shipped over 2.5 billion ARM's core-based microprocessors by January 2005.

Unlike several other vendors, they do not provide free chip documentation. Only the reference manuals are provided, which are mostly sufficient in the vast majority of circumstances. Because individual chip makers are the in charge of chip development, so they can provide some information.

Table 1 shows some key data comparison of the architectures.

	x86	**PowerPC**	**MIPS**	**ARM**	**Cell**
Instruction Set	CISC	RISC	RISC	RISC	RISC
Max. Clock Rate	> 3.2 GHz	2.7 GHz	1 GHz	1.25 GHz	> 3.2 GHz
Implementation	16 - 64 bit	32 - 64 bit	32 - 64 bit	32 bit	64 bit (PPE) 128 bit (SPE)
Endianess	little	big *)	arbitrary *)	little *)	arbitrary *)

*) *switchable*

Table 2.1. CPU architectures for Embedded Systems

With increasing need for technology and their applications, embedded systems have become more and more complex. Easy and simple microcontroller circuits are easy to program, using either an assembly language or a simple language such as C. But these approaches have limited functionality once we want to use advanced features such as multitasking. The implementation of advanced featured by human hand can be tedious and prone to errors. The efficient solution for this is to use an operating system, which makes it easy and favorable to access such features. Linux is one of the commonly used operating systems for Embedded Systems. The history of Linux is well known in the computer history. The name Linux can be used interchangeably with Linux Kernel, Linux Distribution or Linux system. Linux stands only for kernel. Any such system can be built using customizations, and can also be based on present binary distributions, an example of which is Debian. Linux was developed as a Posix conform gpos. Linux serves as a multi-user system, which fits well for any application type, mostly multi-functional. Hence it is a big system, which needs multiple resources from memory to processing power, and other scheduler requirements. A large amount of memory which is used in desktop distribution include desktop utilities, documents, etc., and are a removable option when it comes to embedded targets. It is a great possibility to successfully create a linux system which is completely functional by using a very small amount of memory. The kernel offers high customization options, and it is also an option to remove kernel functionalities that are not required. Linux is readily available for both 32 and 64 bit architectures. Three software costs are available in traditional embedded systems. The initial development setup is the cost for purchasing the license from the vendor for the OS. The cost for additional tools includes the tools which are given by the developer package, in case they turn out to be insufficient. And the runtime royalties include the cost per unit when the system is deployed. All development tool and OS Components are readily available for Linux, free of cost whereas the license prevents the royalties.

2.3 Embedded Linux

The main difference between Linux and Embedded Linux is found in its configuration. Where a normal Linux is more generic, Embedded Linux is usually aligned with a hardware in combination to the application for which it is intended. A generic Linux kernel can run across multiple devices and machines, and embedded linux kernel can have issues in the same.

Linux uses an architecture in protected mode, with the help of memory implementation in memory mode in Intel processors from 80386. The processor is capable of operating using four privilege levels. Any program running at the highest level, level 0 has all privileges such as I/O instructions, interrupt enabling and disabling and many more. Lower levels, on the other hand, prevent programs from carrying out many operations. In the context of Linux, Level 0 refers to kernel space and level 3 refers to the user Space.

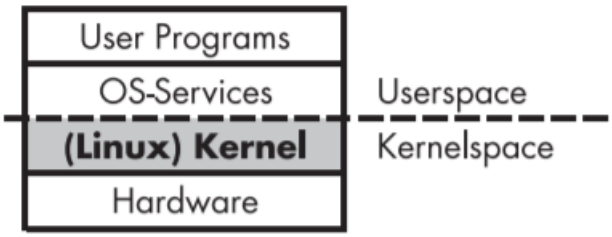

Figure 5. Structure of UNIX

Two shown layers are further divided into two more layers. Hardware controllers are present in the bottom layer. It is a combination of all physical devices in a UNIX or linux installation. The next layer is the kernel, and its purpose is to abstract and provide access to the hardware resources, even the CPU. The user layer consists of the OS-services and the user applications as well. The operating system has all the OS-services. The programming interface for the kernel, is a part of this sub-system.

Linux kernel can be referred to as a monolithic kernel. It runs only in the kernel space in the supervisor mode. It provides a high level of virtual interface, by which

computer’s hardware can be accessed. The use of the kernel is computer resource management, and the provision for programs to run and make use of those resources. The resources offered include the CPU, the memory and the input-output. CPU is the heart of the system, responsible for the execution of all programs. The responsibility of making the decision that which running process has to be assigned belongs to the kernel. A scheduling algorithm is used by the kernel for selection of tasks, as the linux system works on pre-emptive multitasking.

Instructions and data is stored in the memory. For a program to execute successfully, data and set of instructions both should be present inside the memory. The kernel decides which memory any process is allowed to use, and how to proceed if there is not enough memory available. Any input output present as a part of the system, including disks, displays etc. are a responsibility of the kernel too. The kernel is the deciding partner on allocating application requests to perform any input-output operation for the requested device. The kernel's C-Library provides routines for the kernel's system calls, which allow programmes to use the services offered by the kernel. These system calls are used to call all kernel functions. Inter-process communication, or IPC, is another feature of the kernel that allows for synchronization and communication between processes.

At runtime, executable kernel modules can be loaded and unloaded dynamically. As a result, when needed, the kernel's capabilities can be simply extended. Hardware modules, for example, might be loaded exactly before the hardware is needed.

When compared to code that is directly built into the kernel, kernel modules introduce a tiny overhead. Despite this, only loading modules when they are required helps to reduce the amount of code executing in kernel space. Kernel modules can usually only be loaded into kernels for which they were designed. In order to create a stable Linux kernel, you must have the sources for all kernel modules required by your system.

Some hardware manufacturers only provide closed-source drivers, making it difficult

to upgrade or to create a kernel that is custom-made for the system. Pre-made kernel modules are notorious for causing instability, but there is a kernel feature that allows you to load modules designed for a different kernel.

2.4 Benchmarking

Benchmark stands for the condition where several programs or tests, with the motive to assess the hardware as well as the software of the system. Benchmark testing can also be put as a scientific method for testing the performance indicator for the different, but same type of object to be tested. Hardware benchmarking is different from software benchmarking. Hardware benchmarking evaluates parameters such as the processor performance, graphics card, memory card, the network and the hard disk. Two types of benchmarking are present: application and synthetic benchmarking. Application benchmarking is used to evaluate the performance of real-time applications, including servers and databases. On the other hand, synthetic benchmarking includes putting stress on a component, which can include reading or writing data. The purpose of benchmarking i.MX 8M is to weigh its performance against the previous processor, i.MX 53. The CPU performance needs to be checked using benchmarking, to confirm whether applications for the products can run on the new processor and are compatible with minimal compromises in performance. Multiple tools are available for benchmarking linux systems. These tools are available under open-source license and can be used easily. Some of these tools include SysBench, 7z, IOzone, HardInfo and many more. We have used SysBench tool for our work. SysBench is a cross-platform, modular, multi-threaded tool for benchmarking various OS parameters which are necessary for any system which runs a database under the influence of wide and extensive database . The SysBench tool helps to gather information about the system performance, without the need of setting up any complex database benchmark. It is also possible to run this benchmark without installing any database. The SysBench model enables to test various system parameters such as File I/O performance, the

scheduler performance, the memory allocation and transfer speed, POSIX threads implementation performance and database server performance. SysBench runs over a very simple design. It executes a fixed number of threads which execute all requests simultaneously in parallel format. Different test modes produce different levels of workload, hence the workload produced is dependent on the test mode running. The total number of requests or the total time for which the benchmark runs, or even both when the situation demands. The test modes which are available use compiled-in modules when being implemented. Every test mode has additional customizable option available. Any new test modes can also be added according to need. The version used in our work is 1.0.18.

The general syntax for Sysbench is shown as follows:

```
sysbench [common-options] --test=name [test-options] command
```

Figure 6. General Sysbench Syntax

Some commonly used commands and their meanings are given below.

Prepare: This command carries out the necessary actions which are needed in few tests in the preparation step. For example, it helps in creating the required files on the disk space for the file io test, the database filling during oltp test.

Run: It runs the actual test using the –test=name option.

Cleanup: When certain tests use the prepare option to create files for testing, the cleanup command helps removing those temporary files to free up space.

Help: This command can provide with the available options for any test, using –test=name option.

Option	*Description*	*Default value*
--num-threads	The total number of worker threads to create	1
--max-requests	Limit for total number of requests. 0 means unlimited	10000
--max-time	Limit for total execution time in seconds. 0 (default) means unlimited	0
--forced-shutdown	Amount of time to wait after --max-time before forcing shutdown. The value can be either an absolute number of seconds or as a percentage of the --max-time value by specifying a number of percents followed by the '%' sign. "off" (the default value) means that no forced shutdown will be performed.	off
--thread-stack-size	Size of stack for each thread	32K
--init-rng	Specifies if random numbers generator should be initialized from timer before the test start	off
--test	Name of the test mode to run	*Required*
--debug	Print more debug info	off
--validate	Perform validation of test results where possible	off
--help	Print help on general syntax or on a test mode specified with --test, and exit	off
--verbosity	Verbosity level (0 - only critical messages, 5 - debug)	4
--percentile	SysBench measures execution times for all processed requests to display statistical information like minimal, average and maximum execution time. For most benchmarks it is also useful to know a request execution time value matching some percentile (e.g. 95% percentile means we should drop 5% of the most long requests and choose the maximal value from the remaining ones). This option allows to specify a percentile rank of query execution times to count	95

Table 2.2. Common command line options

In some circumstances, having not just the end benchmark statistics, but also frequent

dumps of current stats to see how they change through the test run is beneficial. SysBench supports a batch execution mode for this purpose, which is enabled with the --batch option. The delay between consequent dumps can be expressed using seconds, using the –batch-delay provision.

Example, sysbench –batch –batch-delay=5 –test-threads run.

This will start SysBench in threads test mode, and every 5 seconds, the current minimum, average, maximum, and percentile values for request execution times will be reported.

The various test modes available in sysbench are as follows:

1. CPU: The cpu is among SysBench's most basic benchmarks. Each request in this mode consists of computing prime numbers up to the value supplied by the --cpu-max-primes option. All calculations are done in 64-bit mode. Each thread conducts requests in parallel until the overall number of requests or total execution time exceeds the restrictions set by the common command line arguments. Example, sysbench –test=cpu –cpu-max-prime=20000 run

2. Threads: This test mode is created to measure the performance of schedulers, specifically when a scheduler has a high number of threads contending for a set of mutexes. SysBench produces a certain number of threads and mutexes based on the number of threads and mutexes given. Next, every thread begins processing the requests by locking the mutex, yielding the CPU so that the scheduler can place the thread in the run queue, and then unlocking the mutex when the thread is rescheduled to execution. The following steps are repeated numerous times in a loop for each request, and the more iterations are completed, the more concurrency is imposed on each mutex. –threads-yield and –thread-lock options are provided in this test mode. –threads-yield option is used to set the number of locks loops which are to be executed per request.

3. Mutex: This test mode was created to simulate a situation in which all threads run concurrently for the most of the time and the mutex lock is only held for a short duration (incrementing a global variable). The motive behind the benchmark is the examination of the mutex implementation performance. The default value for the actual mutex lock to be randomly chosen before each lock is 4096.

4. Memory: This test mode is useful for evaluating sequential memory reads and writes. For all memory operations, each thread can access either a global or a local block, depending on command line settings. The size of memory block to be used can be customised with this benchmark, the default value is 1K. The total size of data to be transferred can be defined, with both read and write operations available.

5. Fileio: This test mode may generates a wide range of file I/O workloads. SysBench produces a specified number of files with a specified total size during the prepare stage, and then each thread performs specified I/O during the run stage. This group of files has been subjected to operations.

CHAPTER 3
PRODUCTS BASED ON EMBEDDED LINUX

Neptune3, Sonopet, Core2 and Multigen are products developed on the base of Embedded Linux.

3.1 Neptune3

Neptune3 is a medical device, that falls under the hood of Waste Management System. During any surgery, hazardous bodily fluids and smoke are complimentary. They can prove to be very dangerous for the surgeon's as well as the patient's overall health and well-being. There are times during a surgery, when cotton heavily soaked in blood is mistaken for blood clots and may be left inside the human body after the completion of the surgery. Such cases have been reported in the past, where the patient comes back later with complaints. Neptune3, a successor to its previous versions, was developed with the motive to avoid and completely remove the possibility of such situations. The Neptune3 Waste Management system was successfully developed after five years of research and development. It is a multi-tasking device, used by surgeons all around the world, providing them with considerable safety and efficiency. Neptune3 has successfully moved surgical suction to a completely new level.

The product has the availability of the Hush Vacuum Pump. As the name suggests, this pump provides a quieter fluid suction system. The current version of the product produces a noise level of just 4.3 sones, which is up to 48% quieter than its predecessor, the Neptune2. A household ventilating fan can produce up to 7 sones of noise, making the Neptune3 even quieter than it.

Figure 7 depicts the Neptune3 device. It has multiple suction range indicators with two displays. High suction audio alerts with manifold suction range indicators are also available. Integrated smoke evacuator is the standard equipment, with an 80-hour

ULPA filter. Two independent suction cannisters work in the range of 50-520 mm-Hg of suction. Tactile feedback dials are provided for the control of suction limit, which displays on the screen as on-screen numeric and indicators for low-medium-high indicators. The SealShut Technology successfully locks away all the sucked biohazards.

Figure 7. Neptune3

A typical in-use screen of the Neptune3 is shown in figure 8. It depicts the menu which provides the start or stop data of the suction range, the actual mm-Hg, the suction limits available and the total cannister volume with the cannister volume reset.

Figure 8. Neptune3-in use screen

Figure 9 depicts the screen for the built-in smoke evacuator. This built-in evacuator automatically detects the smoke at the surgical site and has the provision of adjusting the flow once the device is switched to auto-mode. The product has the provision of patient-to-patient reuse by just changing the manifold. The presence of internal rotating power washers has provided extra cleaning power. The electromagnetic couplers themselves engage in automated process of limiting the physical touch.

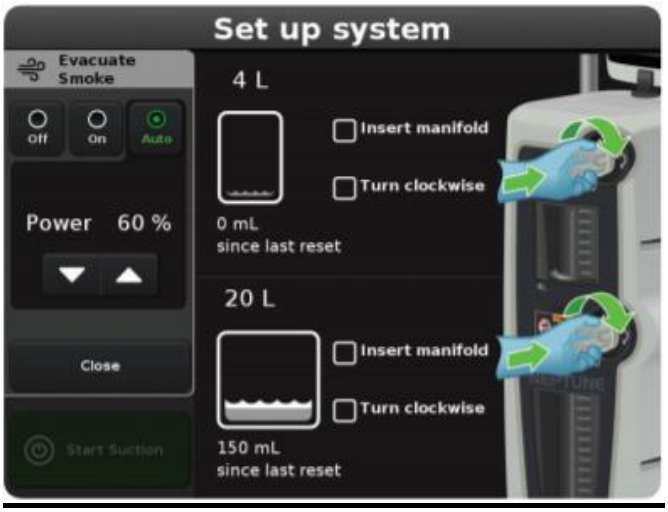

Figure 9. Smoke Evacuator

3.2 Sonopet

The Sonopet is an Ultrasonic Aspirator that delivers Flexibility, for both the surgeon and the patient. It finds it use and applications in the areas of soft-tissue and fine-bone dissection. Since soft-tissue and fine-bone do not require high power for operation, Sonopet offers fine control and dependable power to fragment, emulsify and aspirate soft-tissue and bone.

The Sonopet system has three major components, each designed for maximum performance and reliability. The first component is the hand piece, shown in figure 10. The hand piece is a lightweight, ergonomic product, with angled and straight 25kHz models. It also has the provision of 34kHz, with provides even greater precision. Due to the provision of higher frequency, it gives lower amplitude, that is the distance which the tip moves along its length.

Figure 10. Sonopet Handpiece

The second component, the tip, is shown in figure 11.

With the structure of the product, 20 options are available for the soft-tissue and bone procedures. It is possible to customize the diameters as well as the length of the structure. The several numbers of tips include up to four soft-tissue tips with another five fibrous

tissue tips and remaining 11 as bone tips. Different procedures require different length of the tip according to the bod part that is being operated on. Standard and super long lengths are provided to meet the requirement. The third component, the console, is depicted in figure 12.

Figure 11. Tips

The console is the heart of the device, which is easy to setup and operate as well, with the provision of Automatic Hand piece Recognition. Controlling the power at which suction is performed, the control of irrigation and ultrasonic power are the main functions of the console.

During a complex surgery or procedure, it is crucial for the surgeon to have superior control over the hand piece. Its lightweight nature supports the same. Tactile feedback is another feature present in the system. There is no necessity of an external cooling system, as the system is self-sufficient in the same. Cavitation is the process that helps in the surgeries using the Sonopet system. Negative pressure in the targeted tissue is created using the vibrations of the ultrasonic tip. As the vibrations continue, the cells start to expand, until they cannot anymore, and finally end up bursting. This process of cavitation is selective in nature, as the tissues with higher water content, are more prone to the process.

Figure 12. Console

3.3 Mutligen2

The MultiGen 2 is another medical device which finds its Core in Embedded Linux. The device is responsible for generating radiofrequencies, required for the removal of bodily tissues, a common practice during many surgeries and procedures. This product is an advanced tool especially used by physicians, keeping in mind to cause minimal pain to the patient.

Figure 13. MultiGen2

The MultiGen2 can handle up to 4 lesions at any instance of time, with the added advantage of controlling them independently. The radiofrequencies generated by the device can be used for monopolar as well as bipolar procedures simultaneously. Thermal and pulsed procedures are also carried out at the same time. This differentiation between the procedure or surgery type is an automated process. Compared to its predecessor, the device achieves the target temperature, with minimal errors, usually working at 100Watts. The order of lesions can be selected easily with independently run channels. The device provides the freedom to customize the procedure based on various deciding factors such as the patient needs, the end goals of the procedure and the personal preference. In the Conventional Heat Lesion mode, RF Frequency up to 1MHz, aperiodic damped sinusoid, with deviation of +10% to -10% can be generated. Whereas in the Pulse mode, 1MHz sine wave with selected pulse frequency and on time can be easily generated . The overall time in SI joint procedure or surgery is reduced with the use of larger lesions. Intradiscal procedures, such as spinal procedures, are also supported by the device. The time and temperature step profiles are pre-programmed into the device for intradiscal procedures.

The Multigen2 RF Generator, with the Venom Cannula and Electrode system, can generate less-invasive lesions in a very short period of time.

3.4 Core2

Core2, which stands for Consolidated Operating Room Equipment, Console is also a medical device that has its roots in the Embedded Linux system. This console brings together several power tools, which can be effectively used and controlled using the console itself. It is a versatile device, finding its application areas from orthopedic, dental areas, ENT, neuro and spine areas, to other endoscopic areas as well [9]. The device is capable of powering small as well as large bone drills, saws and drivers, ENT shavers, foot pedals, both small and large joint shavers and bone mill and irrigation [9].

The device is helpful in placing or cutting screws, pins, metal, wires or any other fixation

device [9]. The irrigation flow is customizable and can be set according to the surgery and body part being operated on. Due to the capability of the device to operate a spectrum of power tools and devices, the need of multiple power sources has become redundant.

Figure 14. Core2 Console

There are three hand piece ports available on the system, with the availability of foot pedals as well. Two foot pedal ports are present, with the provision of assigning two pedals to a single hand piece as well. The choice of foot pedals includes NSE footswitch, Bi-directional footswitch and the uni-directional footswitch. Setting the device or instrument settings during an on-going operation/procedure should be avoided as a healthy practice. The device helps to eliminate such disruptions by being able to retrieve all customizations needed for the procedure. These customizations can be set and saved for future purposes. User profiles can also be set and saved. Multiple features can be set and customized on the device including RPM of motor or device, rate of oscillation, direction of movement of hand piece or power tool, break and irrigation control, acceleration settings and so on. The foot pedals can also be customized according to the situation and need. These setting and preferences can be shared with other Core2 consoles over USB cable. Integrated irrigation is set up in the device with one-step

insertion and removal. No additional hardware components are required to affix or tubes to threads. Pole brackets are also available on the device. More room is provided around

the irrigation cassette by using refined port locations. The provision of an internal irrigation pump is the core supplier of coolant for the cutting target, also with adjustable irrigation flow rates. The console is capable of taking the input as 100VAC -230VAC and can successfully drive power-duty hand pieces as well.

CHAPTER 4
EXPERIMENTAL WORK AND RESULTS

All the experimentation and tests were run on an evaluation board. The evaluation board used in our work is MaaXBoard, by NXP. The MaaXBoard is a low-cost board based on the i.MX 8M processor. The family of i.MX 8M processors are based on Arm Cortex A-53 and Cortex-M4 cores. Fig 15 depicts the top view of the board.

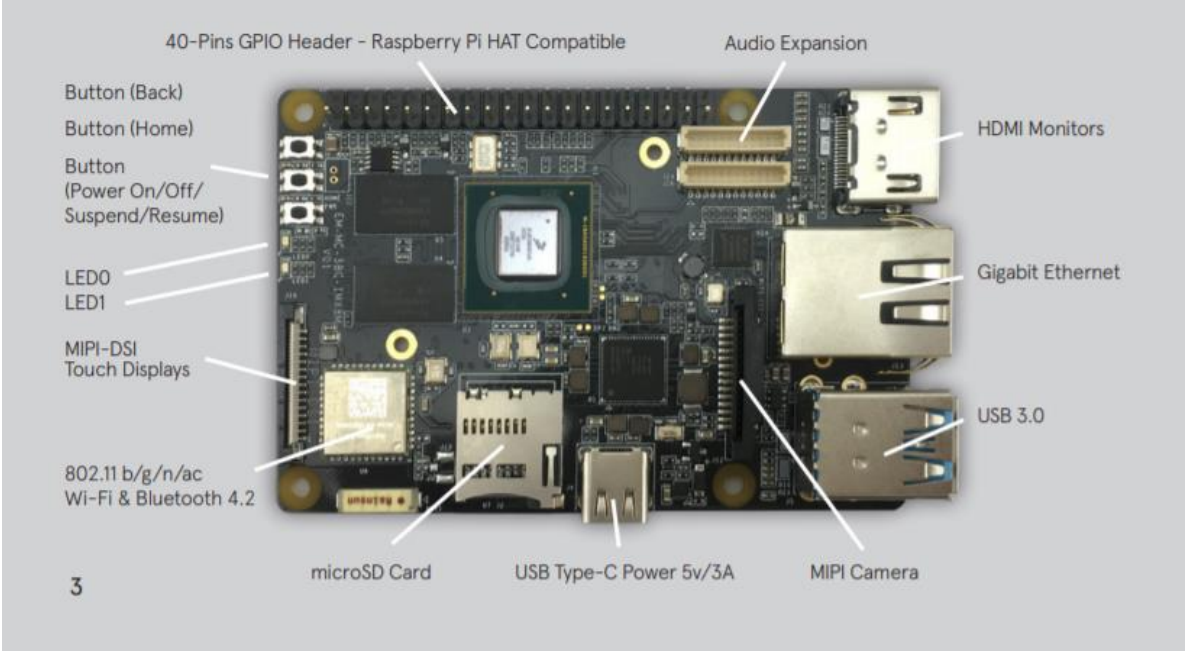

Figure 15. MaaXBoard Top View

The board has a 40-pin expansion interface. 2GB DDR4 SDRAM is present on board, with 8GB eMMC and a Micro SD slot. Multiple options are available for communications and user interface including Gigabit Ethernet, HDMI, Wi-Fi, Bluetooth and more.

A block diagram showing the major components of the board is shown in figure 16.

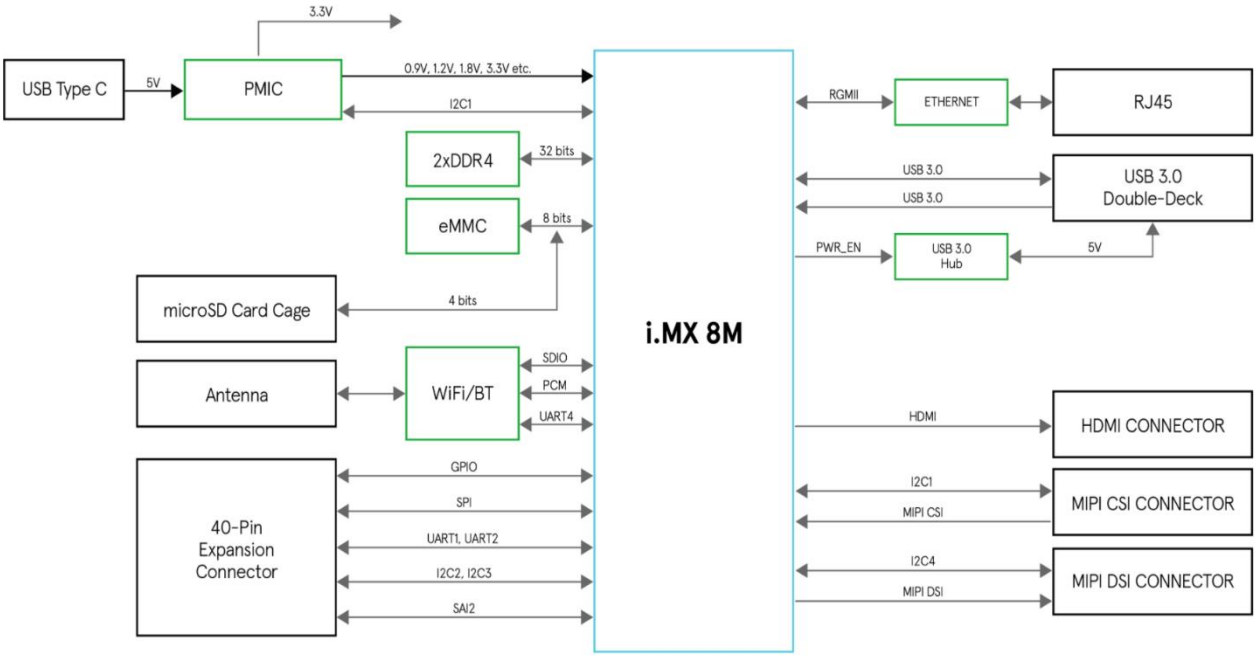

Figure 16. MaaXBoard Block Diagram

The board was powered up using type-C cable (5V/3A) and connecting it to a power port. Since all products and requirements find its core in Embedded Linux, the board was run over a linux system. A Debian image was flashed on the SD Card to boot up the board. The Debian image was a RAR file, which was converted to an ISO file, that is a disc image file, with the motive to be flashed on the SD Card. The software, Balena Etcher was used to burn the ISO image file to the SD Card, shown in figure 17.

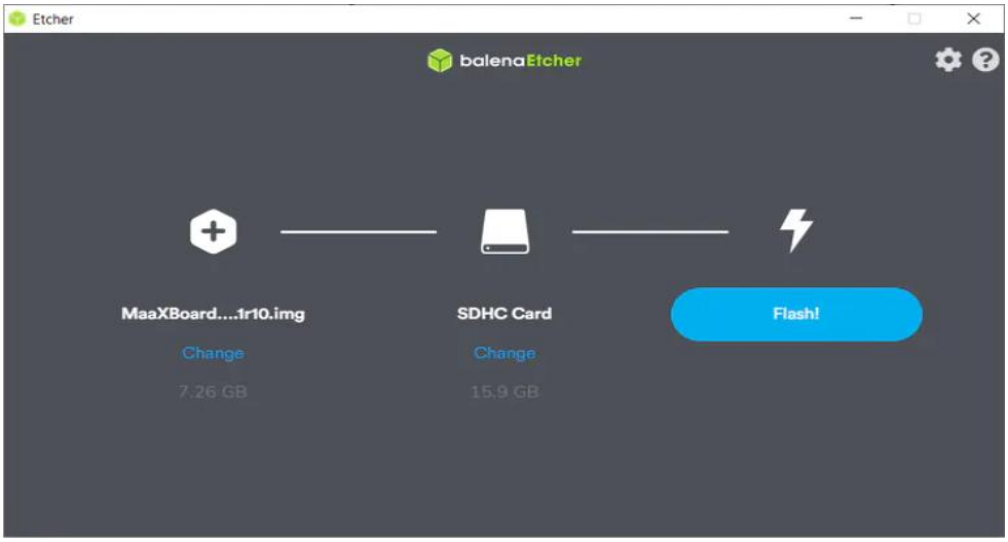

Figure 17. Burning image via Balena Etcher

The board has two LEDs, the system LED, and the user LED. Three buttons are also present on the board, which are S2(power button), S3(back button) and S4(home button). Before powering up the board, it was connected to a monitor using the HDMI port and cable. When the board was powered up, LED0 was lit up, green in color, indicating that the board is connected to power and the Bootloader, that is U-Boot is successfully loaded. As the board boots up, LED1 starts to blink, and multiple character lines are printed on the screen.

Another method was used to check board bring-up with SD Card. This was done by using the serial software, Putty. The following settings were made, as shown in figure 18.

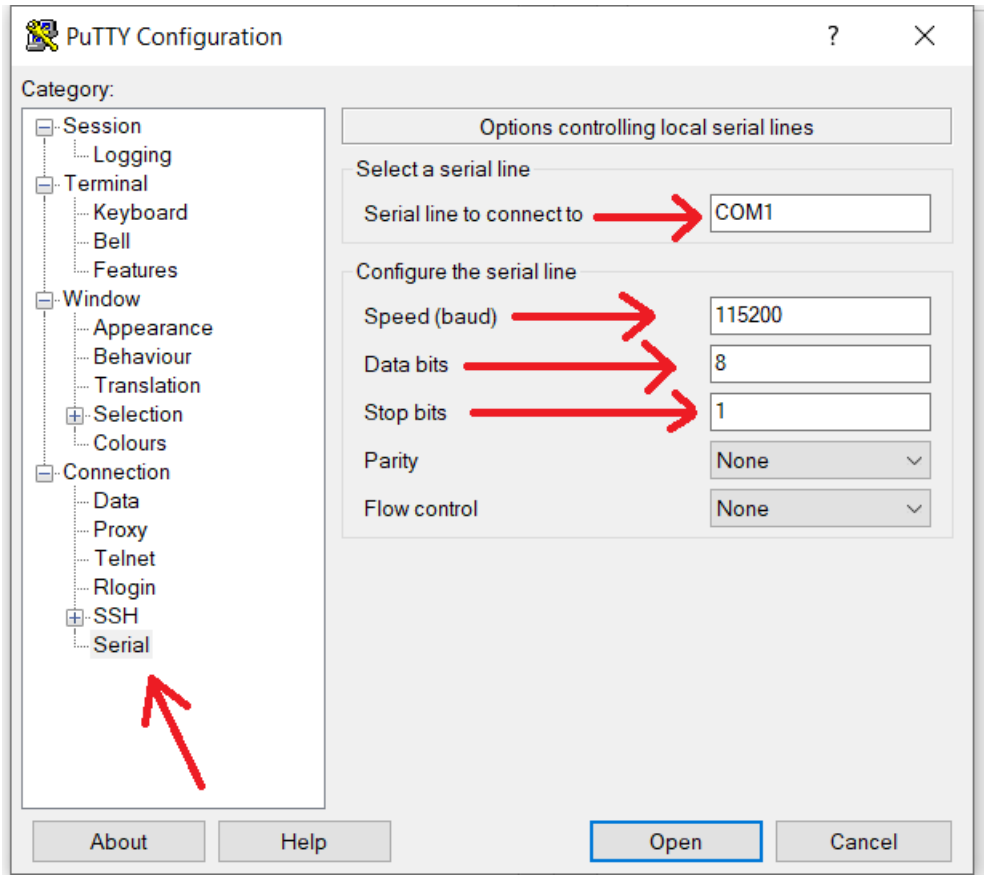

Figure 18. Serial Putty Settings

The Connection type was selected as SSH, ‘Serial’ connection. The correct port was

selected, the baud rate for the communication was set as 115200 bps, and the number of stops bits was selected as 1. The debug pins were connected to the PC, and the board was connected to it using a USB-to-TTL connector. Pin number 6,8 and 10 of base plate J10 on the board correspond to GND, UART1_TXD and UART1_RXD. Pin 8 was connected to RXD of the USB-to-TTL cable, and Pin 10 was connected to the TXD of the same cable. Both the grounds were shorted together as shown in figure 19.

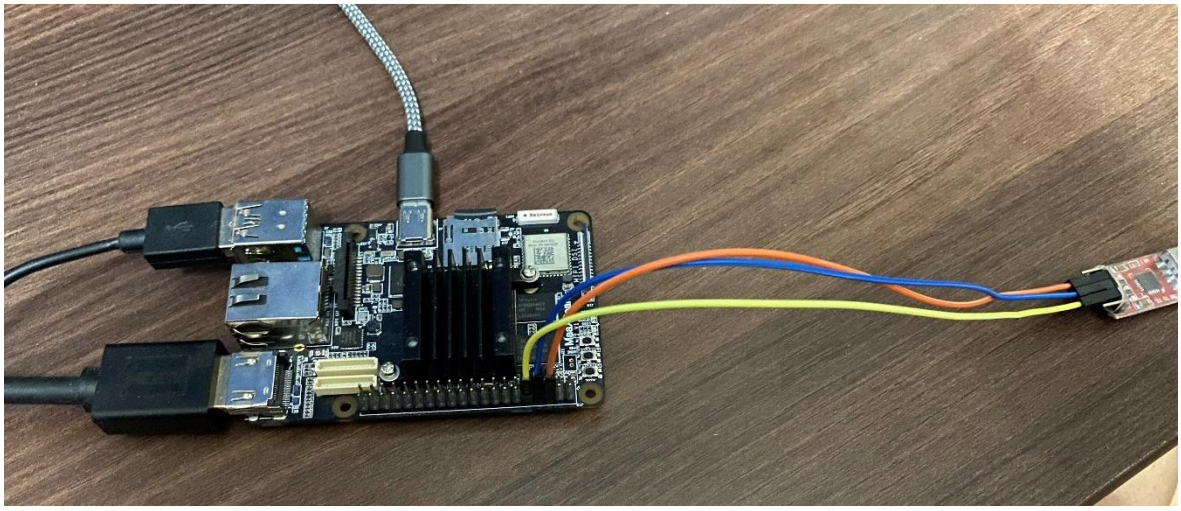

Figure 19. Connections for Boot-up

The SD Card was inserted into the slot, that is J19, and then the board was powered up with a type-C cable of 5V/3A. On booting the board using serial communication, the board displayed the following:

```
NXP i.MX Release Distro 4.14-sumo imx8mqevk ttymxc0

imx8mqevk login:
```

Figure 20. Login after boot-up

The login ‘root’ was used and the password ‘avnet’ was used. The demo application was installed automatically on booting using serial port, and the board restarted automatically after displaying the following.

```
NXP i.MX Release Distro 4.14-sumo imx8mqevk ttymxc0

imx8mqevk login: root
system will install the demo application automatically
tar: embest/bin/qtfm: time stamp 2019-03-04 02:28:33 is 275.130689584 s in the
tar: embest/bin/Chromium: time stamp 2019-03-04 03:01:04 is 2226.130232864 s in
tar: embest/bin: time stamp 2019-03-04 02:59:23 is 2125.130101104 s in the futu
tar: embest/icon/backgroud.jpg: time stamp 2019-03-04 02:31:11 is 432.998416819
tar: embest/icon/32x32/camera.png: time stamp 2019-03-04 03:00:08 is 2169.99277
tar: embest/icon/32x32/utilities-terminal.png: time stamp 2019-03-04 03:00:08 i
tar: embest/icon/32x32/browser.png: time stamp 2019-03-04 03:00:08 is 2169.9896
tar: embest/icon/32x32/wireless.png: time stamp 2019-03-04 03:00:08 is 2169.988
tar: embest/icon/32x32/file-manager.png: time stamp 2019-03-04 03:00:08 is 2169
tar: embest/icon/32x32/video-x-generic.png: time stamp 2019-03-04 03:00:08 is 2
tar: embest/icon/24x24/camera.png: time stamp 2019-03-04 02:36:53 is 774.982419
tar: embest/icon/24x24/browser.png: time stamp 2019-03-04 02:36:53 is 774.98217
tar: embest/icon/24x24/wireless.png: time stamp 2019-03-04 02:36:53 is 774.9817
tar: embest/icon/24x24/file-manager.png: time stamp 2019-03-04 02:36:53 is 774.
tar: embest/icon/24x24/video-x-generic.png: time stamp 2019-03-04 02:36:53 is 7
tar: embest/icon/24x24/chrome.png: time stamp 2019-03-04 02:33:41 is 582.981321
tar: embest/icon/24x24: time stamp 2019-03-04 02:36:53 is 774.981176298 s in th
system will reboot the system to start the demo application
```

Figure 21. Board Start-up

The board was now ready to use.

The two user LEDs present on the board were tested first. The tests were carried out using command line. LED0 is the user-led, and LED1 is the system-led. To turn LED0 on and off, the following commands were given.

```
root@imx8mqevk:~# echo 0 > /sys/class/leds/usr_led/brightness
root@imx8mqevk:~# echo 0 > /sys/class/leds/sys_led/brightness
```

Figure 22. Turning LEDs OFF

```
root@imx8mqevk:~# echo 1 > /sys/class/leds/usr_led/brightness
root@imx8mqevk:~# echo 1 > /sys/class/leds/sys_led/brightness
```

Figure 23. Turning LEDs ON

The board supports multiple communication protocols such as I2C, SPI and UART. UART test was carried out using command line. Before checking for UART communication, pin 16 and pin 18 on plate J10 were shorted together using a male-to-

male jumper wire. Pin 16 corresponds to the Receiver of UART_2 and pin 18 corresponds to the Transmitter of UART_2. These need to be shorted for effective loopback of data. The UART test was carried out using the command line as shown in figure 23.

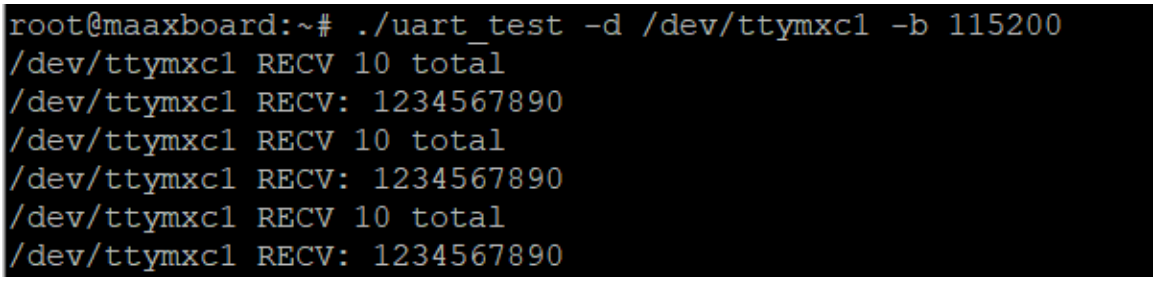

Figure 24. UART Test with Baud Rate 115200

The baud rate used to check UART was taken as 115200 bps. The baud rate for the test was changed to different values, such as 9600 bps and shown in Figure 24, and the same results were achieved.

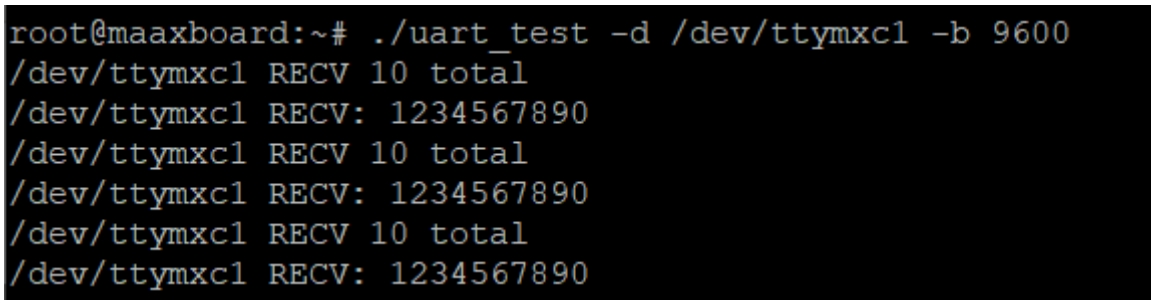

Figure 25. UART Test with Baud Rate 9600

The data, '1234567890' is looped backed effectively from the transmitter pin of the UART module to the receiver pin of the module.

Next, benchmarking of the processor was carried out. Benchmark stands for the condition where several programs or tests, with the motive to assess the hardware as well as the software of the system. Benchmark testing can also be put as a scientific method for testing the performance indicator for the different, but same type of object to be tested. Hardware benchmarking is different from software benchmarking. Hardware benchmarking evaluates parameters such as the processor performance, graphics card, memory card, the network and the hard disk. Two types of benchmarking are present: application and

synthetic benchmarking. Application benchmarking is used to evaluate the performance of real-time applications, including servers and databases. On the other hand, synthetic benchmarking includes putting stress on a component, which can include reading or writing data. The purpose of benchmarking i.MX 8M is to weigh its performance against the previous processor, i.MX 53. The CPU performance needs to be checked using benchmarking, to confirm whether applications for the products can run on the new processor and are compatible with minimal compromises in performance. Multiple tools are available for benchmarking linux systems. These tools are available under open-source license and can be used easily. Some of these tools include SysBench, 7z, IOzone, HardInfo and many more. We have used SysBench tool for our work. SysBench is a cross-platform, modular, multi-threaded tool for benchmarking various OS parameters which are necessary for any system which runs a database under the influence of wide and extensive database. The SysBench tool helps to gather information about the system performance, without the need of setting up any complex database benchmark. It is also possible to run this benchmark without installing any database. The SysBench model enables to test various system parameters such as File I/O performance, the scheduler performance, the memory allocation and transfer speed, POSIX threads implementation performance and database server performance. SysBench runs over a very simple design. It executes a fixed number of threads which execute all requests simultaneously in parallel format Different test modes produce different levels of workload, hence the workload produced is dependent on the test mode running. The total number of requests or the total time for which the benchmark runs, or even both when the situation demands. The test modes which are available use compiled-in modules when being implemented. Every test mode has additional customizable option available. Any new test modes can also be added according to need.

The Linux image flashed on our evaluation board was a Debian image, and hence the following steps were used to install SysBench on the system:

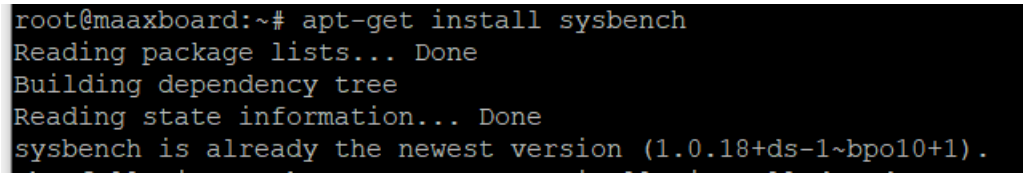

Figure 26. Installing Sysbench

The version of the sysbench software installed was as shown in the below figure:

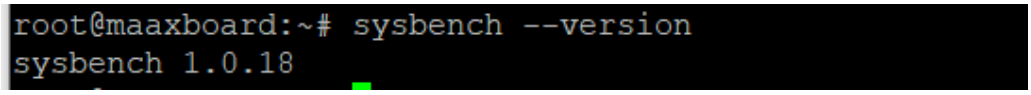

Figure 27. Sysbench Version

Version 1.0.18 has been used for our work. The general syntax of commands used in sysbench follow the following format:

```
sysbench [common-options] --test=name [test-options] command
```

Figure 28. General Syntax used for Sysbench

Some common commands available in SysBench include ‘prepare’, which carries out the preparative actions for the tests that require the same. Examples can include creation of test files which are needed for the fileio test, the filling of test databases for the oltp test and so on. Another command ‘run’ is used to begin the actual test with the –test=*name* option. The ‘cleanup’ command helps in the removal of the temporary test files and data which is created during running of any test. ‘Help’ command shows all the possibilities or usage information for any available test [17]. The first test run for the work was the CPU benchmarking test. It was carried out using the following command:

```
root@maaxboard:~# sysbench --test=cpu --cpu-max-prime=20000 run
```

Figure 29. CPU Benchmark Test

The purpose of CPU Benchmark command is to put full load on the CPU, and test its performance under high load. To achieve the same, what SysBench does is that it produces the specified number (for our case=20000 prime numbers). Then these prime numbers are

verified by performing a standard division of the numbers between 2 and the square root of the same number. On doing so, if the remainder produced is zero, the next number is then calculated. This is done to put more and more stress on the CPU, in order to check its performance. Our test results for CPU benchmarking are shown in figure 29.

```
root@maaxboard:~# sysbench --test=cpu --cpu-max-prime=20000 run
WARNING: the --test option is deprecated. You can pass a script name or path on
the command line without any options.
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   296.20

General statistics:
    total time:                          10.0021s
    total number of events:              2964

Latency (ms):
         min:                                    3.35
         avg:                                    3.37
         max:                                    3.47
         95th percentile:                        3.43
         sum:                                 9998.96

Threads fairness:
    events (avg/stddev):           2964.0000/0.00
    execution time (avg/stddev):   9.9990/0.00
```

Figure 30. CPU benchmark Results for 20000 nos.

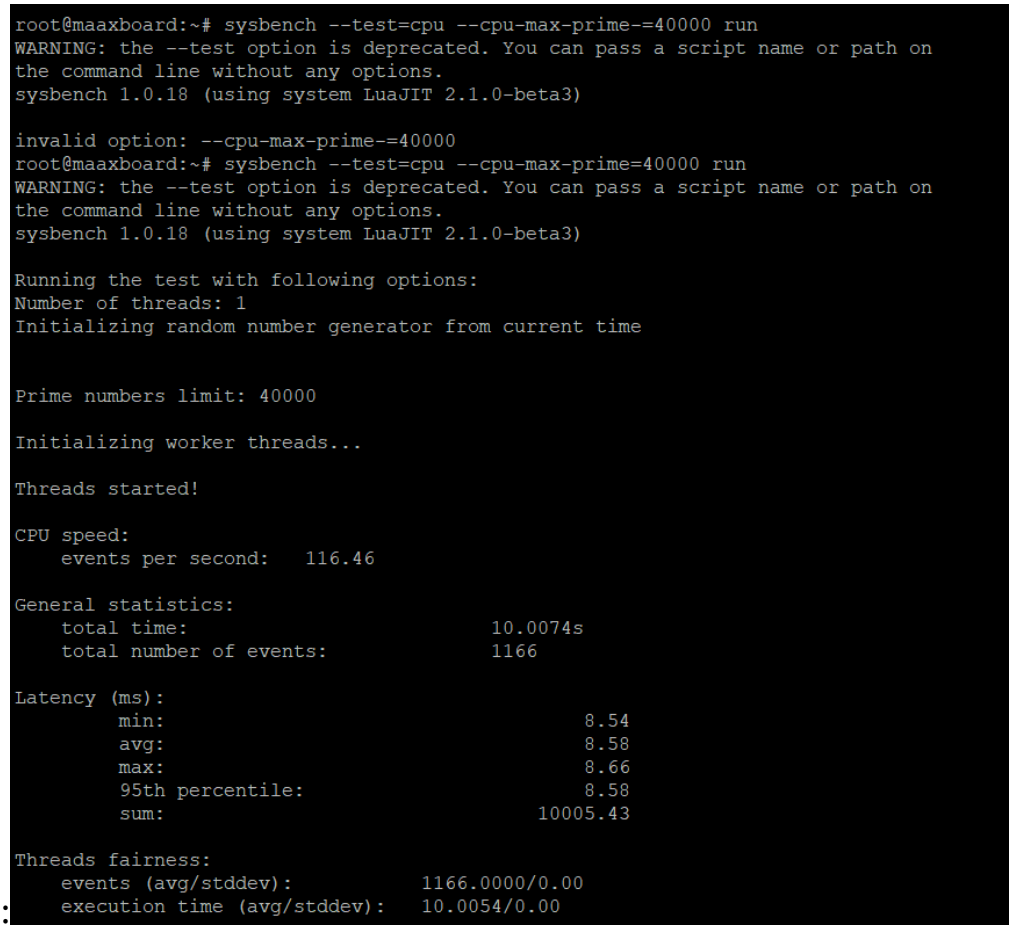

Figure 31. CPU Benchmarking for 40000 nos.

On an average, generating and checking CPU performance with 20000 prime numbers takes upto 3.37 ms, whereas this average time increases when 40000 numbers are generated and goes upto 8.58 ms. A lower average time means the CPU takes lesser amount of time to reach the limit.

For multithreaded applications, another test was run with the appropriate number of --threads as per need.

```
root@maaxboard:~# sysbench cpu run --threads=8 --time=60 --cpu-max-prime=20000
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 8
Initializing random number generator from current time


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:  1189.44

General statistics:
    total time:                          60.0038s
    total number of events:              71376

Latency (ms):
         min:                                    3.35
         avg:                                    6.72
         max:                                   31.47
         95th percentile:                       15.55
         sum:                                479898.92

Threads fairness:
    events (avg/stddev):           8922.0000/16.54
    execution time (avg/stddev):   59.9874/0.01
```

Figure 32. CPU Benchmark with 8 threads in use

Here, the number of events increase as we have used 8 threads. --cpu-max-prime is a measure of the work done per event, which means that a single event calculates the prime numbers up to the specified limit.

Memory was tested next. For memory tests, we can test both read and write speeds. Either of the two mode can selected using –memory open. Sequential and random access modes can also be selected using –memory-access-mode. Memory benchmark also has a special stop condition –memory-total-size, which can be used for the test. Depending on the command line, every thread has the option of accessing either a local or global block for all the memory operations involved. The test results for Memory workload are shown in figure 32.

```
root@maaxboard:~# sysbench --test=memory --num-threads=4 run
WARNING: the --test option is deprecated. You can pass a script name or path on
the command line without any options.
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 4
Initializing random number generator from current time


Running memory speed test with the following options:
  block size: 1KiB
  total size: 102400MiB
  operation: write
  scope: global

Initializing worker threads...

Threads started!

Total operations: 49385981 (4936120.19 per second)

48228.50 MiB transferred (4820.43 MiB/sec)


General statistics:
    total time:                          10.0002s
    total number of events:              49385981

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                    0.75
         95th percentile:                        0.00
         sum:                                18769.18

Threads fairness:
    events (avg/stddev):           12346495.2500/5179.77
    execution time (avg/stddev):   4.6923/0.00
```

Figure 33. Memory Workload Test results

When the memory workload test is performed, the benchmark application that is Sysbench, allocates a memory buffer from which it can either read or write to, every time equal to the size of the pointer, which is either 64-bit for our test, and continues for every execution until the total specified buffer size has been read from or written into. This process is repeated until the specified volume given by –memory-total-size is reached. The important result obtained from is the throughput and the number of operations per second, which is 49385981 (4936120.19 per second) for our processor.

The next test carried out was the fileio standard test. This benchmarking is carried out in three steps: The first step is creating test files for the benchmark, here we create 128 test

files. We have used 5Gb of space for our work. The next step is to run the benchmark using the test files and display the results. The last step includes removing all the test files which have been used already. The three steps are shown below:

```
sysbench --test=fileio --file-total-size=128G prepare
sysbench --test=fileio --file-total-size=128G --file-test-mode=rndrw --max-time=300 --max-requests=0 run
sysbench --test=fileio --file-total-size=128G cleanup
```

Figure 34. Steps for fileio benchmark

Step 1:

```
Creating file test_file.122
Creating file test_file.123
Creating file test_file.124
Creating file test_file.125
Creating file test_file.126
Creating file test_file.127
5368709120 bytes written in 480.91 seconds (10.65 MiB/sec).
```

Figure 35. Creating Test files

Step 2:

```
root@maaxboard:~# sysbench --num-threads=16 --test=fileio --file-total-size=5G --file-test-mode=rndrw run
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 16
Initializing random number generator from current time


Extra file open flags: (none)
128 files, 40MiB each
5GiB total file size
Block size 16KiB
Number of IO requests: 0
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      176.92
    writes/s:                     117.37
    fsyncs/s:                     562.79

Throughput:
    read, MiB/s:                  2.76
    written, MiB/s:               1.83

General statistics:
    total time:                          10.4568s
    total number of events:              6919

Latency (ms):
         min:                                    0.00
         avg:                                   23.16
         max:                                  463.35
         95th percentile:                      110.66
         sum:                               160252.70

Threads fairness:
```

Figure 36. Fileio Benchmarking

```
Threads fairness:
    events (avg/stddev):           432.4375/25.01
    execution time (avg/stddev):   10.0158/0.01

root@maaxboard:~# sysbench --num-threads=16 --test=fileio --file-total-size=5G --file-test-mode=rndrw cleanup
WARNING: the --test option is deprecated. You can pass a script name or path on the command line without any options.
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Removing test files...
```

Figure 37. Removing and clearing Test files

The type of workload which has to be run can be chosen from sequential read, write, or random read and write, or a combination of both. In our work, we have used random read and write, rndrw. The time duration of the test was given using –max-time. The benchmark provides information about reads/s and writes/s which are 176.92 and 117.37 for our test. The read and write throughput are 2.76MB/s and 1.83MB/s.

Another benchmark carried out was Threads benchmark. This benchmark is carried out to check the scheduler performance, specially under the case when a large number of threads compete for the same set of mutexes. A specific number of threads and mutexes are created. Next, every threads starts to run the request which includes locking the mutex, CPU yielding, hence placing the thread in the run queue by the scheduler, and finally unlocking the mutex once the threads get rescheduled for execution. For every request, these steps repeat multiple times in a loop in order to perform more iterations, and hence more concurrency is put on each mutex.

The threads test and its results are shown in figure 37 below:

```
root@maaxboard:~# sysbench --num-threads=64 --test=threads --thread-yields=100 -
-thread-locks=2 run
WARNING: the --test option is deprecated. You can pass a script name or path on
the command line without any options.
WARNING: --num-threads is deprecated, use --threads instead
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 64
Initializing random number generator from current time


Initializing worker threads...

Threads started!


General statistics:
    total time:                          10.0374s
    total number of events:              7712

Latency (ms):
         min:                                    0.14
         avg:                                   83.14
         max:                                  879.34
         95th percentile:                      325.98
         sum:                                641146.97

Threads fairness:
    events (avg/stddev):           120.5000/12.47
    execution time (avg/stddev):   10.0179/0.01
```

Figure 38. Threads Benchmark with 64 threads

Total time of 10.03 seconds was approximately consumed, with an average latency of 83.14ms. When the number of threads were increased from 64 to 128, the following results were obtained:

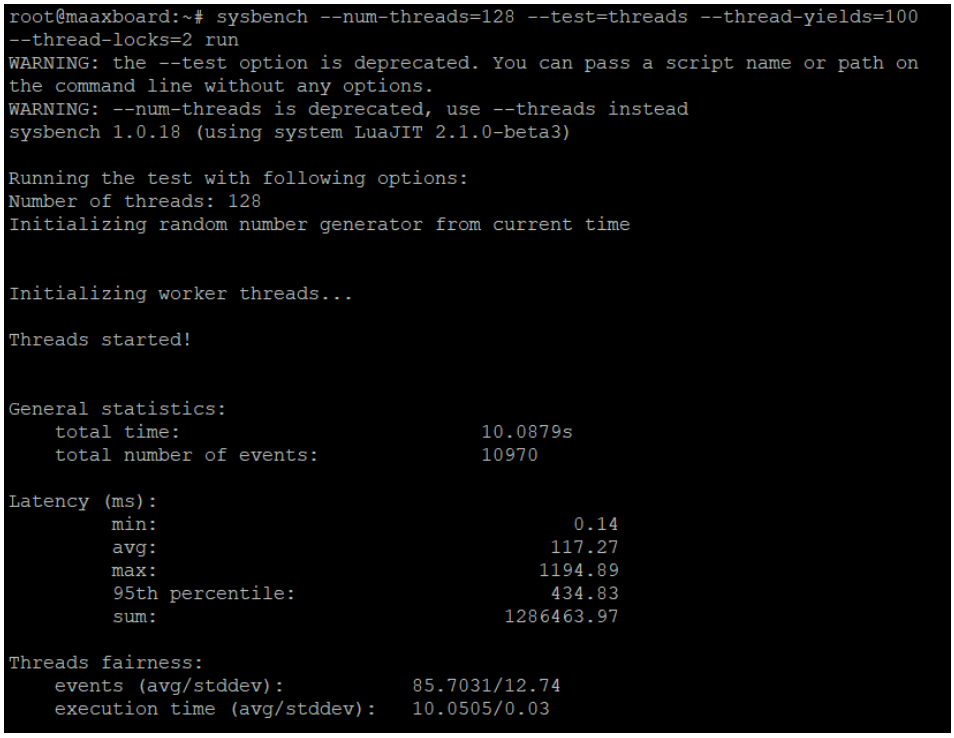

Figure 39. Threads Benchmark with 128 threads

From both the threads benchmarks, we can see the total time and latency increase up to 10.0879 secs and 117.27ms.

Next we carried out the Mutex benchmark. When this benchmark is carries out, a situation is emulated where all threads run concurrently, and acquire a mutex lock for a small/short period of time.

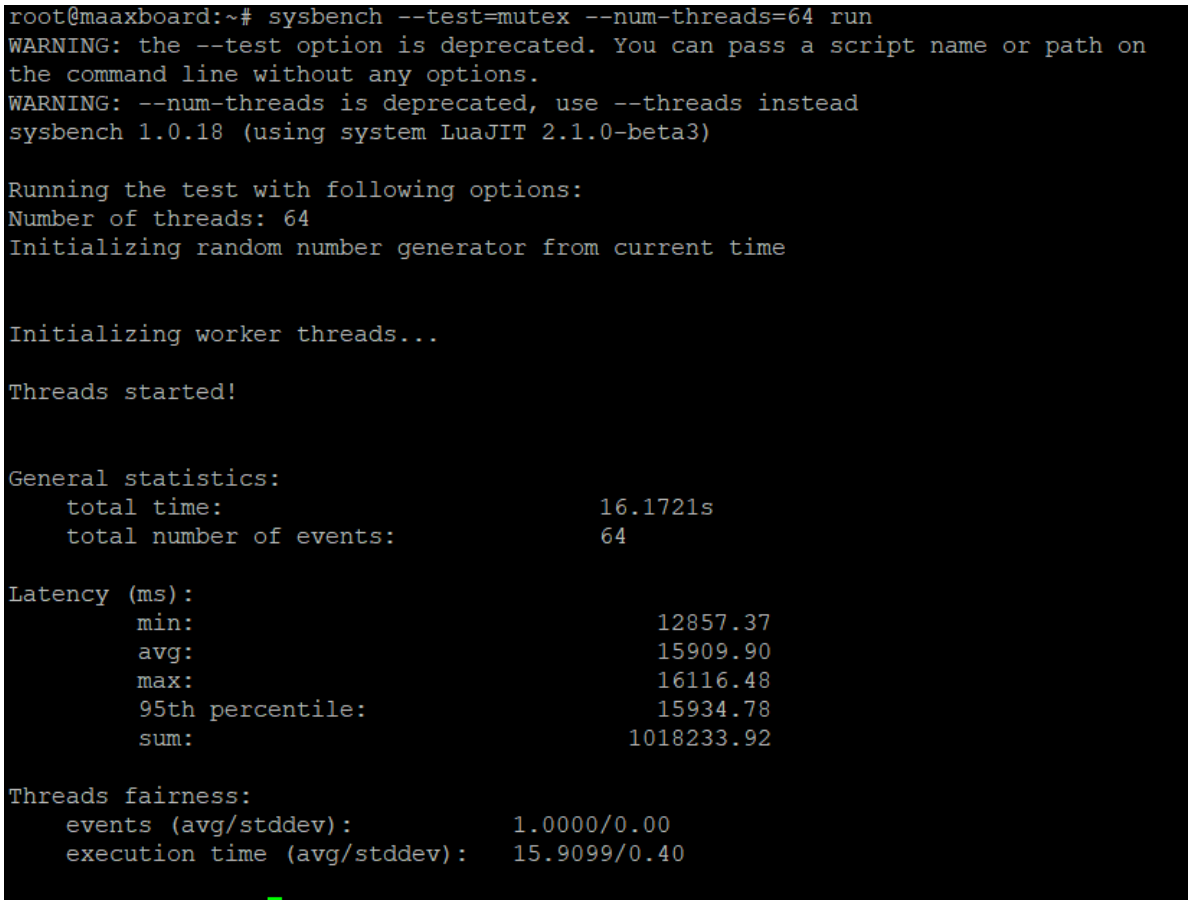

Figure 40. Mutex Benchmarking

Mutex implementation was tested, with 64 threads running. The performance of mutex implementation is examined using this benchmark. Under the mutex workload, sysbench runs a single request per thread. The benchmark first puts stress on the CPU by the use of an incremental loop, by the –mutex-loops parameter. Then it uses a random mutex, or lock, increments the value of the global parameter and then releases the lock. These steps are repeated multiple times up to the limit of the number of locks --mutex-locks.

After all the benchmarking was complete, the software bundle for one of the products, Neptune was ported to the evaluation board. The following steps were followed to make the system ready for running the Neptune application:

Step 1: Install the necessary packages using Apt-get install command

Apt-get install build-essential libc6-dev libtool sharutils libncurses5-dev libgmp3-dev

libmpfr-dev gawk gettext bison flex gperf indent texinfo libgtk2.0-dev libgtk2.0-bin libsdl1.2-dev swig python-dev texlive-latex3 texlive-extra-utils binutils-dev automake guile-1.8 icon-naming-utils libdbus-glib-1-dev wget gtk-doc-tools libxml-parser-perl zip unzip ecj fastjar x11-xkb-utils libglade2-dev libperl-dev python-libxml2 libexpat1-dev gconf2 groff libc6-dev

Step 2: run dpkg-reconfigure dash and respond "No" to the prompt asking "Install dash as /bin/sh?"

Step 3: Once the base packages are involved, main screen string generation utility and volume screen string generation utility are made and installed.

Step 4: chmod 777 .config command was used to get read and write permissions and everything was built successfully.

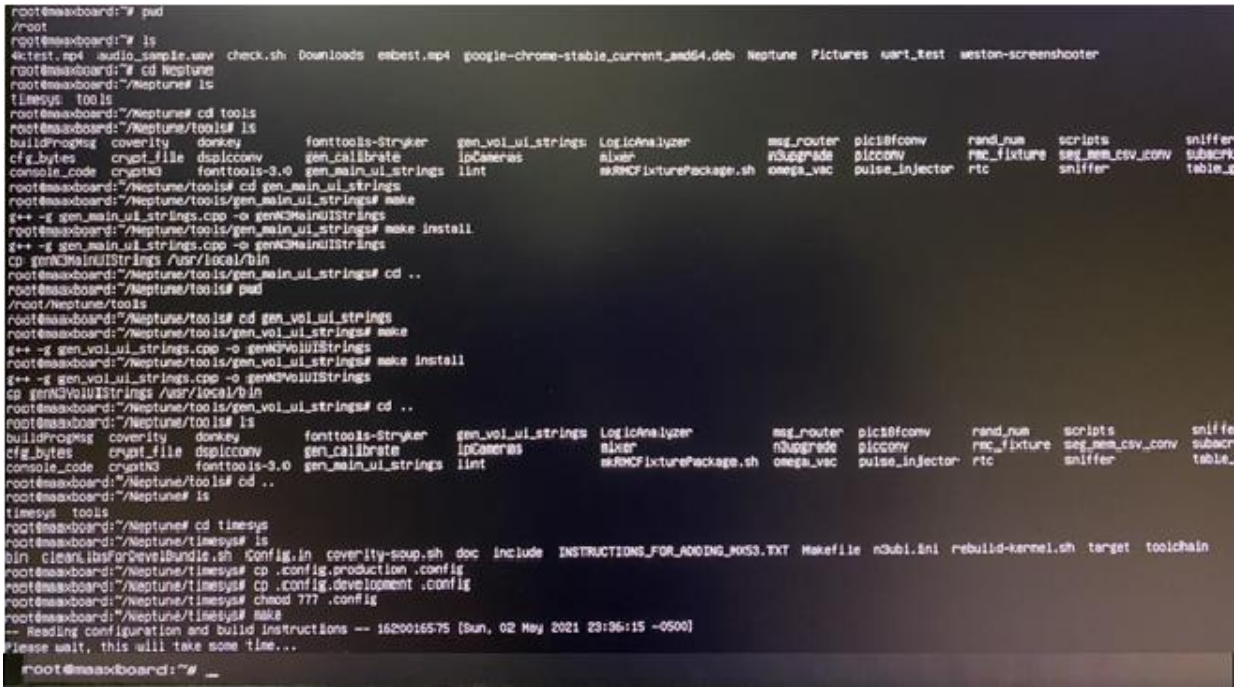

Figure 41. Build Successful

The system was successfully built. As the system was successfully built over the new

processor, POC, Proof of Concept was successful. The modules can be switched over the new processor with the necessary pin mapping changes.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

With the aim of moving the SOM board over to a new processor, an evaluation board with the new processor, that is i.MX8 was used. The board was successfully brought up and basic communication and LED testing was carried out. The main communication protocol used is UART. Rigorous testing of the evaluation board was done, and the processor was benchmarked. All the benchmarking tests were successful and the software for one of the existing products was pushed on the board, and the build was successful. For future works, pin mapping will be carried out from the old processor, i.MX53 to the new one. The kernel might need some customization changes when moving to the new processor. U-Boot will also be built on the same path. The software for other products will also be checked for successful build on the evaluation board, and moved with necessary pin mapping.

REFERENCES

[1]https://www.mddionline.com/software/6-advantages-linux-medical-device-applications

[2] https://www.embeddedcomputing.com/technology/open-source/linux-freertos-related/using-linux-in-medical-devices

[3] M. Jin, X. Zhou, P. Duan, Z. Tian and J. Zhou, "The Design and Implementation of Embedded Configuration Software Based on Embedded-Linux," 2008 International Conference on Computer Science and Software Engineering, Wuhan, China, 2008, pp. 98-101, doi: 10.1109/CSSE.2008.640.

[4] 9100-003-550-Rev-A_SI160646-N3-4-Page-Slick-v9_pdf_stryker

[5] Weinger, Matthew, et al. "Handbook of Human Factors in Medical Device Design." CRC Press, 2011. PrintM. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[6]Stryker-Sonopet-brochure-catalog: https://www.stryker.com/content/dam/stryker/navigation/products/sonopet/resources/SONOPET-catalog-brochure.pdf

[7] Stryker – MultiGen2-catalog : http://necod.com.ar/catalogoPdf/7/i.pdf

[8] Stryker – MultiGen2-manual : http://orthosurgical.es/1/upload/multigen_2.pdf

[9] https://neurosurgical.stryker.com/products/core-2-console/

[10] Core2 console brochure - https://www.stryker.com/content/dam/stryker/nav-nse/products/Core-2/resources/Core%202%20Brochure.pdf10

[11]http://www.wemed1.com/downloads/dl/file/id/6434/product/1118/manual_for_or_s5400_50.pdf - Manual

[12] MaaXBoard Hardware user manual

[13]https://www.prevas.dk/download/18.58aaa49815ce6321a327da/1506087244328/Yocto_Debian_Whitepaper.pdf

[14] A. G. Moya and Á. B. Barros, "Practical course of embedded Linux for XUPV2P development boards," 2012 Technologies Applied to Electronics Teaching (TAEE), Vigo, Spain, 2012, pp. 98-102, doi: 10.1109/TAEE.2012.6235416.

[15] P. Radhavan, A. Lad, S. Neelakandan: “Embedded Linux System – Design and Development”, Auerbach Pub. 2006.

[16] Z. Jiang, "A Linux Server Operating System's Performance Comparison Using Lmbench," 2016 International Conference on Network and Information Systems for Computers (ICNISC), Wuhan, China, 2016, pp. 160-164, doi: 10.1109/ICNISC.2016.043.

[17] https://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf