ISSN:0975-9646

Operating System on Microcontroller in Context of Emulator

Sneha.M.L[#1], Dr. Rohini Nagapadma[*2]

[#1]*PG student, Department of E&E,*
The National Institute of Engineering, Mysuru, India
[*2]*Professor, Department of E&C,*
The National Institute of Engineering, Mysuru, India

***Abstract*— A common question asked by most embedded engineers is "can GNU/Linux run on a microcontroller?" The answer to that is rather simple; it can be done with ease if we have a paged memory management unit and a few megabytes of main memory .There are quite a few methods, which can be adopted, in order to run GNU/Linux on a microcontroller, among which, the most advantageous method is picked by us. The method used is to run GNU/Linux within the context of an emulator.**

***Keywords*— Emulation, Microcontroller, Abstraction Layer, IoT, uCLinux, Linux, Boot loader, Booting.**

I. Introduction

We live in an amazingly high-tech world, surrounded by electronic gadgets, which are mainly controlled by microcontrollers. The situation we find ourselves today in the field of microcontrollers had its beginnings in the development of technology of integrated circuits. This development has enabled us to store hundreds of thousands of transistors into one chip. That was a precondition for the manufacture of microprocessors. The first computers were made by adding external peripherals such as memory, input/output lines, timers and others to it. Further increasing of package density resulted in creating an integrated circuit which contained both processor and peripherals. That is how the first chip containing a microcomputer later known as a microcontroller has developed. The reason we find microcontrollers fascinating is that they have been and continue to be such an important part of the electronics industry. Over the past decade more microcontrollers have been creeping into our daily lives. In today's world, microcontrollers are used in just about every electronic object in the household and place of business. Just about the only common object in the house that does not have a microcontroller in it is the light bulb. In fifteen years or so even that may not be the case. The reason microcontrollers have become so common is that they are more than merely reliable. By adding a small computer to any devices it is possible to increase efficiency and safety. Timing devices are now composed almost entirely of microcontrollers. This has made them unbelievably accurate. They are also cheap, and much more reliable. A digital watch today, which has no moving parts, is almost impossible to break through normal use. It is easy to adapt digital watches to extreme environments such as the deep sea or vacuum. Telephones and other personal communication devices also use microcontrollers. With such devices it is possible to have wireless telephones and cellular phones, each capable of maintaining a connection between the phone unit and some sort of base station. These phones can also encrypt data as it leaves and decrypt it as it comes in. Televisions and stereos use microcontrollers. Neither is the mess of tubes that was synonymous with televisions and radios. As a result, both produce better quality picture and sound, have more features, and weigh less per unit volume. All kinds of transportation system use microcontrollers. Cars use them in fuel injection systems, brakes, airbags, and just about any other piece of equipment. Airplanes are going to a "fly-by-wire" control system. This is a complex computer interface between the controls that the pilot uses and the control surfaces of the plane. Such interfaces are controlled by microcontrollers. Hence microcontrollers find a predominant role in automobile and industrial applications.

There are wide verities of microcontrollers available in the market today, for any application you choose. The code written for one microcontroller cannot be executed on any other microcontroller. In that case you have two ways to deal with the situation i.e. you can either make the code platform independent or see that your microcontroller can execute any damn code you want it to execute.

The former method just means that your microcontroller should be capable enough to execute any code you put on it. How is it possible that a microcontroller will be capable enough to execute any program you put on it? It is possible only if you can have a general purpose operating system running on microcontroller.

How can one have a general purpose operating system running on a microcontroller? Is it seriously possible for a measly system as that of microcontroller to run an operating system? There are quite a few methods, which can be adopted, in order to run a general purpose operating system on a microcontroller, among which, the most advantageous method is to run operating system within the context of an emulator. Emulation is what we do when we try to make one system behave like or imitate a different system.

The next question that arises is which operating system to choose among wide verities of general purpose operating system available? The major reason why operating systems are not run on microcontroller is its memory constraint. Hence we will have to choose the operating system wisely such that it occupies comparatively less memory. In the list of operating systems Linux is the one which requires comparatively less memory hence we land up with that. In this project we will be running GNU/Linux version

II. RELATED WORK

[1] This article aims to revive Lisp programming language for native, interactive and incremental microcontroller (MCU) program development by running a dialect of Lisp (PicoLisp) as virtual machine on the target. This article basically gives information about the layers of code involved before having an application code over the microcontroller in the context of virtual machine. Many interesting, practical embedded solutions have been developed so far with such languages, supported as part of a virtual machine (VM). Figure1 shows the system architecture of a natively-programmable, digitally controlled system.

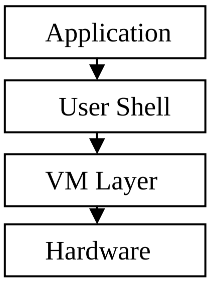


Fig.1 General MCU software system with a VM layer

With the above architecture, it is possible to write abstract, self-adapting, middle-level drivers for hardware modules on the MCU. This enables the possibility of platform-independent, native embedded software development.

The author also highlights that codebase can be made portable across various platforms and architectures simply by using the following key principles:

- Code that is platform-independent is common code and should be written in portable ANSI C as much as possible
- Code that is not generic (mostly peripheral and CPU-specific code) must still be made as portable as possible by using a common interface that must be implemented by all platforms. This interface is called platform interface.
- Platforms vary greatly in capabilities. The platform interface tries to group only common attributes of different platforms.

This article gives a fairly rough idea about how to develop/adopt a virtual Machine to reach to emulate GNU/Linux on a microcontroller.

[2] This book is Cluster of European Research projects on the Internet of Things – CERP-IoT – comprises around 30 major research initiatives, platforms and networks working in the field of identification technologies, such as Radio Frequency Identification and in what could become tomorrow an Internet-connected and inter-connected world of objects. This book basically reports to you about the research and innovation issues at stake and demonstrates approaches and examples of possible solutions.

Closer look to this book will make you realise that the Cluster reflects exactly the ongoing developments towards a future Internet of Things – growing use of Identification technologies, massive deployment of simple and smart devices, increasing connection between objects and systems.

[3] In this white paper author states that The Internet of Things (IoT) is a novel paradigm that is rapidly gaining ground in the scenario of modern wireless telecommunications. Unquestionably, the main strength of the IoT idea is the high impact it will have on several aspects of everyday-life and behaviour of potential users. From the point of view of a private user, the most obvious effects of the IoT introduction will be visible in both working and domestic fields. The basic idea of this concept is the pervasive presence around us of a variety of things or objects – such as Radio-Frequency Identification (RFID) tags, sensors, actuators, mobile phones, etc. – which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbours to reach common goals.

As recently as five years ago, vehicles were merely a means of transportation, but today cars have become the ultimate connected device. By 2020, 90 percent of new cars will be enabled through extensive connectivity platforms. Automobiles that are increasingly intelligent are changing the concept of mobility to consumer-driven preferences that extend beyond the vehicle itself.

As the boundaries of the auto industry blur and as new competitors enter the fray, the traditional industry participants are learning to thrive despite technological disruption. Electronics, telecommunications and insurance companies as well as emerging start-ups are joining the race to find new ways to attract and excite consumers to elevate their experiences with cars.

Connectivity is just the first step in providing a new experience. Many companies can enable connectivity, but just how useful is connectivity without the ability to derive new insight? Many opportunities remain untapped as connectivity and the Internet of Things expands. The key building block comes from volumes of data flowing from one point to another. This data volume remains the most pressing challenge for the auto industry: tapping into this data, combining it with other information and uncovering actionable insights through cloud operations and investment in building new business models that generate value for customers.

- By 2020, the connected car will be the top connected application.
- In 2020, 250 million vehicles will be connected and fully packed with sensor technologies.
- Connected vehicles will produce 350 MB of data/second by 2020.
- One-third of consumer data will be stored in the cloud by 2016.
- In-vehicle software will be updated over the air through cloud connectivity.

Typically, a connected car made after 2010 has a head-unit, in car entertainment unit, in-dash system with a screen from which the operations of the connections can be seen or managed by the driver. Types of functions that can be made include music/audio playing, smartphone apps, navigation, roadside assistance, voice commands, contextual help/offers, parking apps, engine controls and car diagnosis [4].

On January 6, 2014, Google announced the formation of the Open Automotive Alliance (OAA) a global alliance of technology and auto industry leaders committed to bringing the Android platform to cars starting in 2014. The OAA includes Audi, GM, Google, Honda, Hyundai and Nvidia [5]. On March 3, 2014, Apple announced a new system to connect iPhone 5/5c/5S to car infotainment units using iOS 7 to cars via a Lightning connector, called CarPlay Android Auto was announced on June 25, 2014 to provide a way for Android smartphones to connect to car infotainment systems.

III. NEED FOR PLATFORM INDEPENDENCY

The present world of technology, demands for code to be platform independent. As per the present day scenario each microcontroller vendor have their own controller architecture, which means program from another processor vendor can't be executed on our microcontroller, which means it's difficult for the controller users to switch from one vendor to other. In order to meet the present day demand of platform independency we have come up with a simple and cost efficient solution. That is to have an operating system on microcontroller.

IV. EXISTING AND PROPOSED SYSTEM

Any microcontrollers can (in theory) run an operating system which can execute processes and not just threads but they are generally used for application specific requirements. So, most microcontrollers don't come with a memory management unit (which is common for most general purpose processors). The implication is that one can't run GNU/Linux on a microcontroller directly.

To run GNU/Linux on microcontroller we have two methods:

- Interface an SDRAM controller with the MCU (for additional main memory) and modify the Linux kernel to remove MMU specific attributes (like what ucLinux does)
- Run GNU/Linux within the context of an emulator for the target

In the first method mentioned above we will have to modify Linux kernel, such a way that it is MMU independent and not only that a severe restriction will be imposed on the system software (e.g. a port of a system program that invokes functions like `malloc' or `vfork'). One such kernel has all these modifications - uCLinux. In the later one we will emulate OS on microcontroller.

The figure 2 represents the existing and proposed system, the system excluding the emulation layer represents the existing system the system inclusive of emulation layer represents the proposed one. Addition of the emulation layer provides us with platform independency. With that, what one may have is a platform (any processor with the emulator running on it) which can virtually execute any program from another processor vendor. That simply means that by introducing an emulator, we make programs hardware agnostic.

Emulation is certainly slow but if the target processor hosting the emulator is even two times faster the resultant will still run at a satisfactory speed.

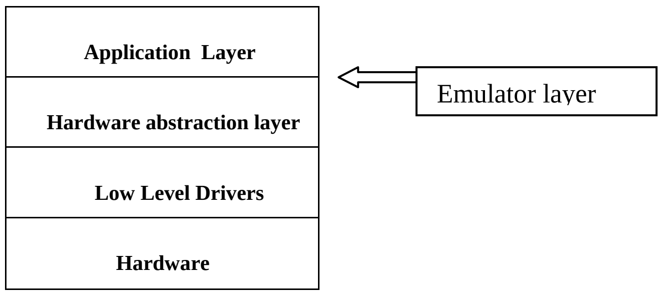


Fig .2 Layered architecture of the proposed and existing system

V. HARDWARE DESIGN DETAILS

As mentioned earlier operating system can be run with ease on microcontroller if we have a paged memory management unit and a few megabytes of main memory. Not all microcontrollers have Megabytes of main memory because microcontrollers are generally used for application specific requirements. So, most microcontrollers don't come with megabytes of memory. In such case what one can do is interface the microcontroller with an external RAM, so that system is provided with sufficient main memory that is required. Not only that, system will also need an external memory to store the operating system and support the microcontroller when it starts booting. For this purpose SD/MMC card can be used. The figure 3 shows how the microcontroller can be interface with both these external memory interfaces.

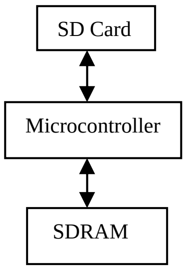


Fig.3 Interconnection of Microcontroller with SD-card and SDRAM

VI. SOFTWARE DESIGN DETAILS

The previous section gives brief description of Hardware requirements of the system as mentioned earlier we will have to interconnect microcontroller with two different external memory units. In order to do that, other than hardware connectivity between them we will have to provide some means of communication protocol. Any kind of serial communication protocol supported by the microcontroller can be used to serve the purpose of interaction between microcontroller and the SD-card. The next problem that has to be dealt with is how to enable communication between SDRAM and microcontroller, this task becomes pretty much easy if we have an EBU (External Bus Unit) on the microcontroller. If we don't have one the other way to deal with this is to have a boot loader which will boot strap the content of the SD-card on to the SDRAM. Once we have established proper communication system between the microcontroller and memory units we will have to think about how to have paged memory system. This is very much required in order to fetch the machine code placed on the SD-card.

To have a paged memory system, we will require a file system layer. After development of file system layer the next step is to develop a Hardware Abstraction Layer (HAL), which will make my system, hardware agnostic. With this it doesn't end, after having all these LLD and HAL we will need a Emulator layer, which will help us in emulating, all the missing functionalities of microprocessor on to our microcontroller (it may be Memory Management Unit, number of bits it can process et..) after having all these layers of software, we are ready to setup the system. Figure 4 represents Interaction between different layers of software.

Once we have all these layers of software and a means to flash the code on to the microcontroller, it just means that we can have Operating System running on Microcontroller.

VII. USE CASE

Having an operating system on a microcontroller just means you will be able to run any damn code on the microcontroller. It just means that the microcontroller can have internet access which means that the microcontroller is capable of interacting with the server and other system surrounding it.

It just means that the electronic instrument which is equipped with this controller can interact with other system without using any cable and it can access internet as well. This plainly directs towards internet of things (IoT).

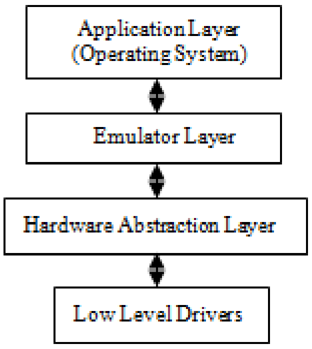

Application Layer (Operating System)
Emulator Layer
Hardware Abstraction Layer
Low Level Drivers

Fig .4 Interaction between Different Layers of Software

The Internet of Things (IoT) is blurring traditional industry boundaries. Automotive channel participants, manufacturers, suppliers, customers, and even vehicles themselves are at the heart of interactions between designers and engineers, production lines, supply chains, and sales and service organizations. With increasingly sophisticated technology, businesses, vehicles, and customers/drivers can now intersect to drive business and personal value across all interaction points and channels, including connected fuelling and convenience offers; parking and toll collection; pay as, how, and where you drive; and traffic avoidance.

IoT is at the heart of this transformation. It connects people, machines, vehicles, parts, and services to streamline the flow of information, enable real-time decisions, and enhance automotive experiences. Leading automotive manufacturers, suppliers, and dealers are already investing heavily in IoT – and realizing returns that range from ultra-efficient inventory management to real-time promotions that grow sales. They are beginning to transform their business practices and recognize that, in time, IoT will touch nearly every area of automotive operations and customer engagement.

The advantages of the IoT vehicle are well-documented, from improved traffic patterns to increased fuel economy. But the largest benefit is safety. The likelihood of collisions will be reduced thanks to DSRC, or Dedicated Short Range Communication (used in V2V and V2I). Of course, this will require huge investments in the public transportation grid. But that also provides a major opportunity to boost the economy with major public infrastructure projects and private partnerships.

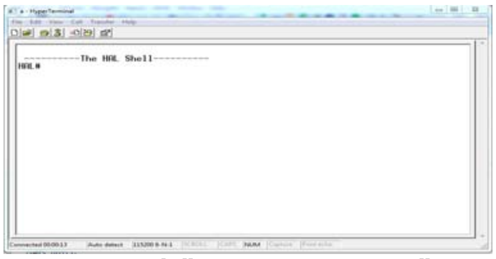
The IoT auto will also be a boon for the electronics industry. The development of better and smaller sensors and smarter real-time data analytics is already underway. The need for automotive-focused security applications and overrides is in its infancy and presents major opportunities for companies willing to dive in. In the consumer electronics market, the entertainment and infotainment industries may also see a boost. (Less time focused on driving means more time open to watching movies or reading eBooks on the highway). Lastly, we see opportunities in the job market itself. A new generation of vehicles will require new forms of education and training for the millions of technicians who will be needed to maintain our future national vehicle fleet.

VIII. IMPLEMENTATION

As long as there's a (cross) compiler, there's really no difference between an emulator, a virtual machine, an operating system or a native compiler. The microcontroller simply does what it is meant to do - execute machine code. To get a UNIX clone (such as an ARM build of GNU/Linux) running with in the context of an emulator on the chip is quite a herculean task. To start with we will have to have all the Low Level Drivers of the microcontroller which we will be using to do this port. Once we have LLds next step is to develop Hardware Abstraction Layer and An Emulator. once you have all the layers of code compile them and flash them on to the microcontroller and invoke the bin file of the kernel placed on the SD card and bootstrap it. After the boot process is complete you will have your Linux shell working on your microcontroller.

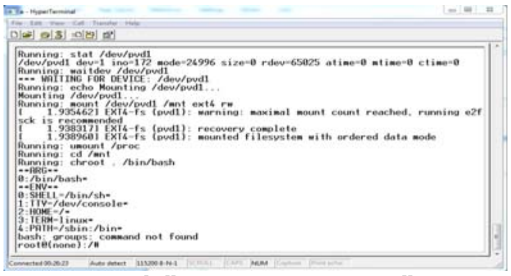
IX. RESULT

In this project we have used HyperTerminal (Terminal emulator) is used to interact with microcontroller and to monitor results. The figure 5 shows the Hardware abstraction Layer shell running on our microcontroller and figure 6 shows the Hardware Linux shell running on our microcontroller.



```
 -----------The HAL Shell-----------
HAL#
```

Fig .5 HAL shell running on microcontroller



```
Running: stat /dev/pvd1
/dev/pvd1 dev=1 ino=172 mode=24996 size=0 rdev=65025 atime=0 mtime=0 ctime=0
Running: waitdev /dev/pvd1
*** WAITING FOR DEVICE: /dev/pvd1
Running: echo Mounting /dev/pvd1...
Mounting /dev/pvd1...
Running: mount /dev/pvd1 /mnt ext4 rw
[    1.935462] EXT4-fs (pvd1): warning: maximal mount count reached, running e2f
sck is recommended
[    1.938317] EXT4-fs (pvd1): recovery complete
[    1.938960] EXT4-fs (pvd1): mounted filesystem with ordered data mode
Running: umount /proc
Running: cd /mnt
Running: chroot . /bin/bash
**ARG**
0:/bin/bash*
**ENV**
0:SHELL=/bin/sh*
1:TTY=/dev/console*
2:HOME=/*
3:TERM=linux*
4:PATH=/sbin:/bin*
bash: groups: command not found
root@(none):/#
```

Fig .6 Linux shell running on microcontroller

X. CONCLUSION

In this white paper we have answered all programmers common question "can GNU/Linux run on a microcontroller?" Not only that we have also explained how a microcontroller can be emulated to work as a microprocessor. This article also proves that any host system can be made to work as any other target system with the power of emulation. This also proves that program can be made hardware agnostic by having different layers of software atop of our hardware.

ACKNOWLEDGMENT

We would like to thank almighty god and our parents, for their constant support. We would also like to thank our friends who providing a healthy environment which helped us complete this work. We also thank the Principal and the Management of NIE for extending support to this work.

REFERENCES

[1] "Programming Mirzar32 Board in Lisp Programming Language" by Raman Gopalan for Electronics For You, April 2016

[2] "Definition of Connected Car – What is the connected car? Defined". *AUTO Connected Car*. South Pasadena, California, United States: A propose. Retrieved 22 July 2014.

[3] Open Automotive Alliance". Open Automotive Alliance. Retrieved 22 July 2014

[4] "Vision and challenges for realising the Internet of Things" edited by Harald Sundmaeker, Patrick Guillemin, Peter Friess and Sylvie Wolfed.

[5] "The Internet of Things: A survey" by Luigi Atzori a, Antonio Iera and Giacomo Morabito for ScienceDirect.