

Customization of GPRS, and Wi-Fi device drivers for PXA270 of Linux OS based barcode scanner

Prabhakar Pujeri, Sanket Dessai

Departement of Computer Science and Engineering, Faculty of Engineering and Technology, M. S. Ramaiah University of Applied Sciences, Bengaluru, India

Article Info

Article history:

Received Nov 19, 2021

Revised Jan 22, 2022

Accepted Feb 10, 2022

Keywords:

EMX-270

GPRS

Linux

PXA270

Wi-Fi

ABSTRACT

To access any device, it is necessary to have an access point. A device driver is an entry point to access a device. This project is aimed to customize the Wi-Fi and general packet radio service (GPRS) device drivers in Linux OS for PXA270 (Intel Xscale ARM processor). Customizing a device driver is a special way of designing software that can be more easily ported from one architecture to another without rewriting it from scratch. The paper is discussing about the customisation of Wi-Fi and GPRS device driver in Linux OS for PXA270 (Intel Xscale ARM processor). To develop a device driver, it is necessary to understand the processor architecture and Linux kernel internals and other design constraints. Since dynamically loaded driver module is attached to the existing kernel, and any error in the driver will crash the entire system. Resource allocation and implementation for a device is one of the main concerns for device driver developers. The device resources are input/output, memory, IRQs and ports. The required toolchain to build the cross-compiler for the Intel Xscale ARM processor was built on Linux platform. The customised device drivers of Wi-Fi, and GPRS was customised, and the customised images are made to port for PXA270 processor architecture on EMX-270 board. With all the supporting parameters the kernel images with drivers are build and ported efficiently. Also, a successful verification and testing had been performed for their functionalities.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Sanket Dessai

Departement of Computer Science and Engineering, Faculty of Engineering and Technology, M. S.

Ramaiah University of Applied Sciences

University House, New BEL Rd, M S R Nagar, Mathikere, Bengaluru, Karnataka 560054, India

Email: sanketdessai0808@gmail.com

1. INTRODUCTION

The two major challenging works in the embedded Linux industry are to write a device driver in Linux and porting Linux operating system on to the target board giving a bigger challenge to the embedded designer, developer, validation, and testing teams. Device drivers play an important file to support the devices in the kernel of the Linux OS. When the application developer uses the device drivers as black boxes where the hardware devices communicate and perform transactions to a defined interface by hiding the details of the working of the device hardware. In this project paper the general packet radio service (GPRS) and Wi-Fi drivers are customisation had been carried out for the PXA270 board for the barcode scanner, where the porting of the Linux for the PXA270 board which is the board design for the barcode scanner along with the soft code designing and porting for the routines of the Wi-Fi and GPRS devices to establish a communication between the client and the outside world [1]–[6].

The reason to choose PXA270 board is a low cost with integrated high end peripherals supporting on the chip and along with the robust processing 32-bit performance of the computer architecture. Linux is supporting the networking features in better and hence it is better suitable for designing the barcode scanner, and it is open source and freeware as compared with the other operating systems. The processor is more suitable based on its Xscale architecture characteristics [7]–[12].

To make the system or any sub-system there is necessary of both the hardware and the software, the software is of the type where the applications and the system software are used and the layer of the firmware which is controlling the hardware through the hardware abstraction layer. Hence there is a need to have a proper synchronisation between the firmware and the software and the hardware, and hence the device drivers are coming into the existence. When a newer development of the hardware takes place and there is necessary for the development of the firmware, software and the application for the developed hardware and the device driver a critical component to be developed for the control, programmability, and usability of the device, hence device driver development and its research are important areas of the research [7]–[12].

Computing architectures are developing and moving into multi-core and hence the Linux operating system or any other operating system need to have supporting APIs for the multi-core at the operating system through the kernel and user level. The solution procedure for this design deals with understanding the PXA architecture and understand its peripherals and develop the device drivers for the networking devices which is the requirement for the barcode scanner for its applications [7]–[12].

2. DESIGN AND IMPLEMENTATION

To perform the porting of Linux on to the PXA270 target board, it is required to perform the build for a toolchain using the cross_compiler, assembler, linker and other standard GNU tools which helps to create and generate an images for the Linux to support the Xscale architecture target board on the host system which is an x 86 PC system [7]–[13]. To build the toolchain, five necessary steps must be followed.

- Binary utilities set-up,
- Kernel headers set-up,
- Bootstrap compiler set-up,
- C library set-up,
- glibc-linuxthreads and, full compiler set-up (which is optional).

The internal of the Linux is open and can be modified by the used which is bigger advantages for the embedded system developer. The embedded system engineers using the standard APIs system called as system calls and other specific APIs at the user or kernel level for the development of the drivers and or customisations of the drivers, sometime the intermediate systemcalls are independent of the architecture are purely based on the software and need to map at the firmware of the system device drivers. Also, the device can be built separately from the kernel and can be plugged in when and when ever required at the runtime either dynamically or can be unplugged when not required. This portability and modularity of the device drivers make the Linux OS and its device driver easy to write for the point and there are hundreds of device drivers are available which are loaded and unloaded for the Linux OS [14]–[19]. This loading and unloading feature make the the Linux based system to grow and the embedded designer's interest to write more and more driver and use the linux for the applications is growing. This growth helps the embedded designs to customisation of the device drivers for their applications as somewhere, someone must write the device driver and make the system work and based on this other designer can customise the device driver for their applications and make the applications work. So without the support of the device driver there is not system functionality will be able to function as today we deal with the co-design of the system where hardware and the software has to work in coordination with each other [20]–[25].

2.1. GPRS and Wi-Fi driver communication

Wireless systems are growing in all walks of the life where people used mobile phones and other applications where wireless components are used as shown in Figure 1. In this application, a barcode scanner needs wireless support. Initially, GPRS was very popular for data transmission over the wireless and now days Wi-Fi plays a significant role in short range and well long range communication along with the support of mobile phones. The wireless applications with the help of Wi-Fi are increasing due to the trends of mobile phones and their components such as bigger screen, processing capabilities and its features. Wi-Fi supporting on the laptop and other computing devices is an emerging alternative for the telephonic lines to convert the voice conversations using the Wi-Fi network for larger applications numbers. The Wi-Fi kind of classes are designed protocol to achive a higher speed internet access and by the standard of IEEE 802.11 b, a, g and so on based on the applications to applications and allowing the compatibility for the devices to connect and

communicate wirelessly where there is not intervention of the physical wiring connections. As the Wi-Fi technology is growing with its speed and characteristics there is more application happening for the betterment of the society where the domain specific language (DSL), TI services and internet access are performed through the Wi-Fi as become a well-established standard [14]–[25].

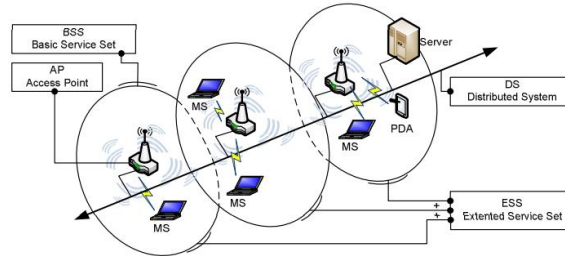


Figure 1. Wi-Fi based on 802.11 infrastructure mode [22]

GPRS is one of the most popular technologies where it is an internet protocol (IP) based services that provide fast and switching based on the packet access to the data network over the internet. The mobile services through the GPRS are improved with the peak-time capacity of a global system for mobile (GSM) network, and hence the GPRS provides packeting access with the external data with high network data transfer with its peak capacity. To support the IP network such as TCP/IP a GPRS is developed where the non-voice services are designed for transmitting the data. The method where the packets are sub-divided as separate packets for the transmission from the smartphone devices and transmit then to the destination in the external network as shown in Figure 2. Any applications based on the internet can run using the GPRS, and the peak throughputs provided are 40 Kbps that is around 53.6 Kbps in raw format. Where there is a sharing among the user who is active in the coverage cell area and hence the throughput will vary based on the active user in the provided cell and its available base-station tower nearest to it.

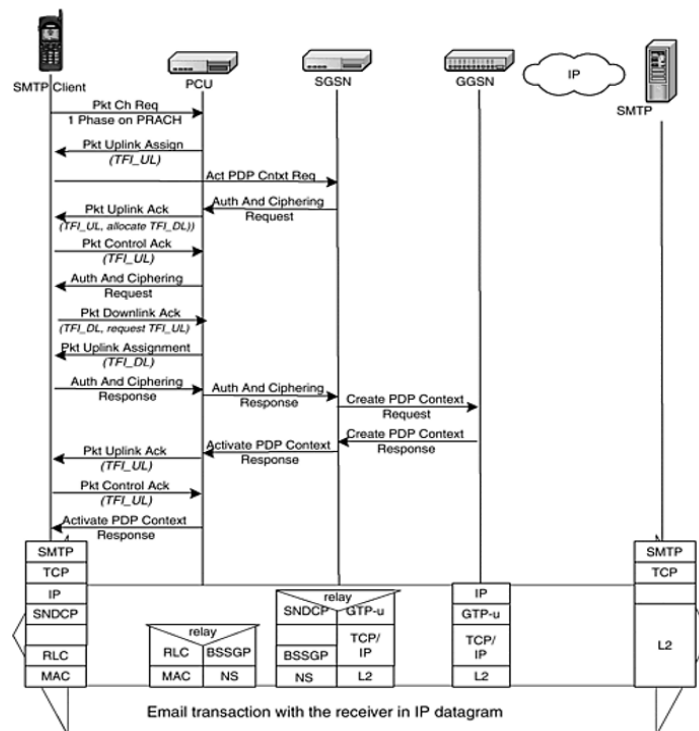


Figure 2. GPRS driver design

Mainly the dynamic IP addressing will be performed by the wireless technologies for data transmission over the static addresses, and the IP manager based systems are developed and implemented for

a wireless dynamic domain name server that is dynamic domain name system (DDNS). IP manager will be tracking the current IP addresses which a modem will be assigning and the DDNS system has three main components which has its functionalities where one of the component functionalities is task to be performed in the firmware of the modem, and issue a notification to the IP manager server based on the new assigned IP addresses.

2.2. Network devices

Each telecommunication to communicate needs to have a network and hence the network layer where mostly the device drivers are written are physical layer medium, which is accessed using the network adapter or the network interface. The physical layer is designed to communicate for the distances based on the technologies, in this application it is more based on the wireless mobile technologies. As shown in Figure 3 to perform distance technologies-based communication interfaces need to support the following properties:

- A through suitable interface between the designed particular hardware for specialized application in the network adapters and the protocol which are software based to communicate for this hardware.
- A protocol stack with its input and output need to support the asynchronous in the kernel of the operating system.
- The different vendors with their network adapters will be implementing with different layers at layer 1 and the layer 2 protocol and hence it is necessary to design and develop to write or customize the piece of software for the different adapters to communicate with the hardware through the network adapter.
- A designer must take care when customization of the driver which is a uniform interface for the access by the protocol instance which is consistent with the applications of the principle of the communications systems layers. Hence based on this layer a suitable implementation to be performed independently for a specific adaptor type.

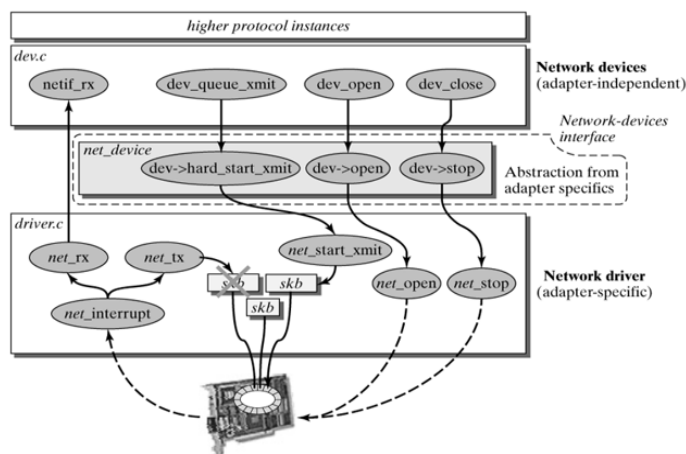


Figure 3. The structure of a network device

2.3. Hardware-specific fields

- Rmem_end, rmem_start, mem_end, mem_start is the field of the memory where the definition at the beginning of the memory, end of the memory spaces, and this memory spaces are shared among the network adapter and the kernel in the Linux operating systems. The location (mem_start? mem_end) is assigned for the buffers for sending the packets and (rmem_start->rmem_end) is assigned for the location for the receiving of the packets. The amount of size required for the buffer is the amount of storage supported by the card and by using the ifconfig the initialization of a network adapter is performed for memory location addresses.
- Base_addr: The search of the device in the firmware and at the driver level for the I/O is performed with its basic address and from these base addresses drivers are memory mapped. To update the values and setting of the values are performed for the display and programmability is achieved by ifconfig. Also, during the loading of the module, the I/O basic address can be specified as a kernel boot parameter.
- Irq: irq is the interrupts, which are numbers based on the services requested in the network adapter. The setting of interrupts is performed for the drivers during the probing phase in which it is defined specifying

the process of when to load as a module into the kernel, or when to start the module in the kernel. To modify the interrupt number `ifconfig` is used.

- DMA: is the direct memory access (DMA) channel having the number of channels which are used by the drivers for temporary storage during the processing. Most 32-bit processor support the DMA.
- Stores: `if_port` stores the media type of the network adapter currently used. For Ethernet, distinguish between BNC, twisted pair (TP), and AUI. There are no unique constants; instead, each driver can use its own values.

2.4. Data on the physical layer

For the Ethernet cards `etherssetup ()` is setting the field values, which are generally or moreless the same identity for all the ethernet-based cards, except for the variations in the flag field, these flag fields need to be set to meet the card capabilities. The APIs and the functions used (`fddi_setup ()`, `tr_setup ()`) to set the standard value for the token-ring and fiber distributed data interface (FDDI) adapters and the fields of the network adapters are set manually as follows:

- The layer 2 packet header length is specified by the `hard_header_length` and for the network adaptor the value is 14. Along with this value the network adapter adding the additional fields which are the preamble and the checksum for the Ethernet and hence this doesn't represent the actual length of the packet header over the physical layer whereas it represents only at the network adapter.
- The maximum length of the payload of a layer-2 frame is performed with the help of the maximum transfer unit (MTU). 1500 bytes are assigned for the MTU for the ethernet and at the Layer-3 protocols must need to consider these MTU values without passing the more octets into the network devices.
- The maximum length of the output queue of the network device is given by `tx_queue_len` and the `ether_setup ()` is setting this value to 100. It is not to be confused between the `tx_queue_len` and the buffers of the network adapter as there is an additional ring buffer for 16 and 32 packets support for the network adapter types in the hardware. These values of the buffer are specified in the request for comments (RFC) 1700 which is specifying the hardware type for the address-resolution.

2.5. Data on the network layer

- The information point of the network devices in layer-3 are given by `ip_ptr`, `ip6_ptr`, `atalk_ptr`, `dn_ptr`, and `ec_ptr`. To configure the parameters of the IP instance then the network device and the internet protocol and its `ip_ptr` point to a structure of the type `in_device`.
- In the case of the IP, this field takes the constant `AF_INET` which are assigned with the family address of the network device.
- `AF_INET` has the length of four bytes for the class of IP addresses and the addresses length of the protocol used is specified by `pa_alen`.
- The addressing of the network layer is described by `pa_addr`, `pa_braddr`, and `pa_mask`. The address of the computer or network device contains at `pa_addr`. `pa_braddr` specifies the broadcast address, and `pa_mask` includes the network mask. All three values are set by `ifconfig` when a network device is activated.
- `pa_dstaddr` specifies the address of the other partner in a point-to-point connection (e.g., point-to-point protocol (PPP) or serial line internet protocol (SLIP)).

A flag includes different switches. Some of them describe the properties of the network device (`IFF_ARP`, `IFF_MULTICAST`); others output the current state (`IFF_UP`). Lists the meaning of these switches, which can be set by use of the `ifconfig` command.

2.6. Device-driver methods

It is necessary to abstract out the network device for the hardware. The methods set which is available for the network drivers functions need to be mapped with the uniform interface so there is accessibility of the higher protocol, and these functionalities are performed with the `net_device` structure implementations. The implementation using the skeleton of the network drivers as an example for the driver development.

- The searching and the initialization of the network devices are performed with `Init ()`. Where the responsibility of the `Init ()` is for finding and initialization of a network adapter. To perform the initialization a `net_devices` structure needs to be created and need to fill with the driver-specified data of

the network devices or the network driver and this registration needs to be performed with the `register_netdevice ()`.

- When the network device needs to be unregistered that is (`unregister_netdevice ()`) for this `uninit ()` is used which is extensive driver-specific functions when the network device is removed. The `uninit ()` is available from the `net_device` structure from version 2.4 and currently not used by any driver.
- When the last reference is removed from the network devices (`dev->refcnt`) a new construct called `destructor ()` is used in the `net_device` structure. This construct is also used to cleanup work which is freeing the memory and hence the `destructor` function is not used by any drivers.
- The activation of the network device is performed by the `open ()` and this activation processes the required system resources assigned. This activation to happen it is necessary to have the device to be registered previously, hence the network device can be used with the successful execution of the `open ()` and hence `ifconfig` command can perform `dev->open ()` is used in the `dev->open ()`.
- To free the network resources, it is necessary to terminate the activities of the network adapter and hence a `stop ()` is used and hence the network device is then no longer active, but it remains in the network registered devices list which is (`net_devs`).
- To check the delivery of the packet over the network device is successful a `hard_start_xmit ()` (in the form of a socket buffer). When the packet was delivered to the adapter successfully then `hard_start_xmit ()` is returned with 0 or otherwise it returns 1.
- To get the information and statistics about the activities about the network devices `get_stats ()` is used, and this information is returned as `net_device_stats` structure. To get the additional information for the wireless network adapter status `get_wireless_stats ()` is used, and this information is forwarded in a structure of the type `iw_statistics`. The tool `iwconfig` can be used to display this specific information.
- `set_multicast_list ()` passed the list with multicast MAC addresses to the network adapter, so that the adapter can receive packets with these addresses. This list is called either when the multicast receipt for the network device is activated (`IFF_MULTICAST` flag) or when the list of group MAC addresses to be received has changed.
- The timing problem during the transmission of a packet across the network adapter is given by the `watchdog_timeo ()`. The kernel calls the `watchdog_timeo ()` method when there is no acknowledgment if the packet is received or not and hence it is time out. `dev->tx_timeout`, to solve the problem.
- When it is necessary to pass the adapter-specific `ioctl ()` commands to the network driver. Then `do_ioctl` is used but this is not used by the higher protocol, due to there is no generic functions category in the device drivers for the network devices.
- To change the network adapter configuration at the runtime a `setconfig ()` is used and the method provide to change the system parameters includes the interrupts and the memory location of the network adapter.

2.7. Managing network devices

The management of the network devices where the knowledge of how the network device is represented by the `net_device` structure in the Linux Kernel. Each `net_device` structure represents one network devices where all the network devices of the Linux kernel need a connection in a linear list in the kernel variable called as `dev_base` which is representing the entry point of the list with the registered network devices. This list help to point the device at the list value of the network device as shown in Figure 4.

2.8. Registering and unregistering network devices

The `dev_base` manages the network devices, and the list stores the activated or unactivated but all registered network devices. When new devices are added with the help of `register_netdevice ()` then the first must create and initialize a `net_device` structure for the `dev_base` and the process is performed with two different ways: It is necessary to specify that in the kernel configuration the driver of the network device is integrated permanently into the kernel and there already available `net_device` structure and with a clear process of the preprocessor definitions it is generated and created different instances of the `net_devices` structure where there is translation and the depending on the kernel configuration and these instance are used for the existing network adapters when there is booting as shown in Figure 4.

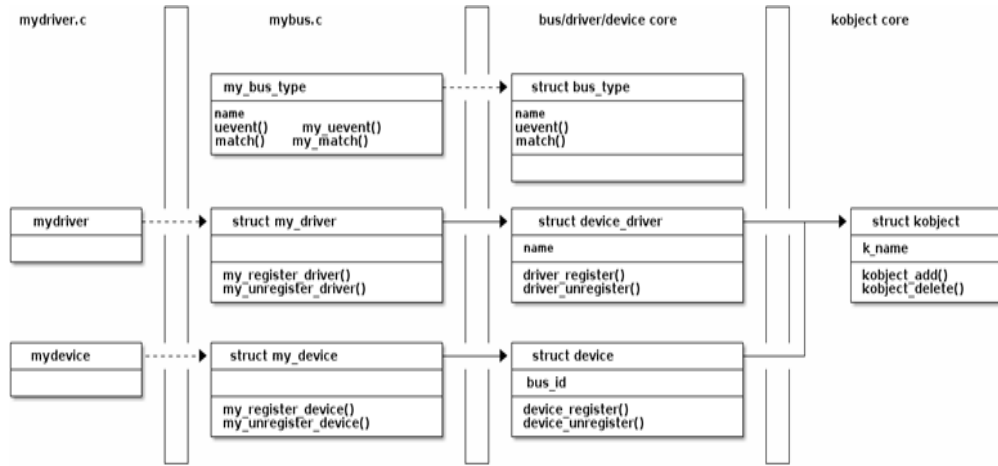


Figure 4. Network driver management [23]

If the driver was translated as a kernel module, then the driver itself must create a `net_device` structure for each existing network adapter. This can be done by the module itself or, for Ethernet drivers, by use of the function `init_etherdev()`.

3. VERIFICATION AND TESTING OF CONFIGURATION AND EXIT LOADER MODULE

The bulk data integration (BDI) is in loader mode when there is no valid firmware loaded or connects to it with the setup tool. While in loader mode, the mode light-emitting diode (LED) is flashing. The BDI will not respond to network requests while in loader mode. To exit loader mode, the "`bdisetup -v -s`" can be used. Also, power-off the BDI, wait some time (1 min.) and power-on it again to exit loader mode. when there is a booting process is configured as shown in Figure 5 and the booting the system for the barcode scanner is taking place as shown in Figure 6.

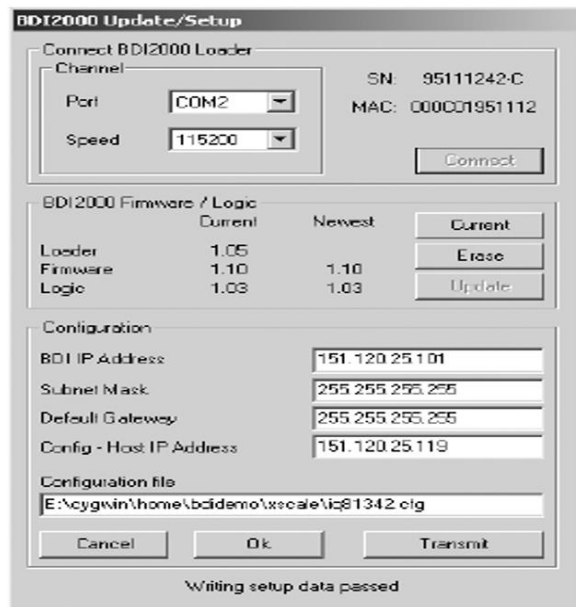


Figure 5. Configure the target using BDI

```

root@prabhakar] ./client 192.168.1.31:2000
successfully connected to server
press ctrl+p to send a data
>123456
press ctrl+p to send a data
>654321
please enter long integer data
received 1456789
data has successfully sent

root@prabhakar] ./server
one client is connected at wifi 10.0.0.2 to 192.168.1.31
press ctrl+p to send a data
received 123456
received 654321
please enter long integer data

```

Figure 6. The Wi-Fi client and server

4. CONCLUSIONS

Device drivers play an important role to support the devices in the kernel of the Linux OS. When the application developer uses the device drivers as black boxes where the hardware devices communicate and perform transactions to a defined interface by hiding the details of the working of the device hardware. The communication and transaction for the data transfer from the system to physical media and then from the physical media to the upper layer of the kernel is performed with the right setting and porting of the Linux onto the PXA270 target board. The developed and customised device driver for Wi-Fi and GPRS are performing as per the expectation with the required performance and without the overhead for the kernel to communicate with the outside world. The Linux kernel had been optimised as per the requirement of the Barcode scanner applications and made an image for the PXA270 board. With this port a different application for the barcode scanner can be added.




REFERENCES

- [1] V. Eswer and S. S. Naik Dessai, "Processor performance metrics analysis and implementation for MIPS using an open source OS," *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 10, no. 2, p. 137, Jul. 2021, doi: 10.11591/ijres.v10.i2.pp137-148.
- [2] V. Eswer and S. S. Naik Dessai, "Embedded software engineering approach to implement BCM 5354 processor performance," *International Journal of Software Engineering and Technologies (IJSET)*, vol. 1, no. 1, p. 41, 2016, doi: 10.11591/ijset.v1i1.4568.
- [3] S. S. Naik Dessai and V. Eswer, "Embedded software testing to determine BCM 5354 processor performance," *International Journal of Software Engineering and Technologies (IJSET)*, vol. 1, no. 3, p. 121, 2016, doi: 10.11591/ijset.v1i3.4577.
- [4] A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. Tripp, *Software engineering body of knowledge*. 2004.
- [5] K. D. Kissell and C. Langgaard, ".../include/asm-mips/mips32_cache.h," *MIPS Technologies*. p. 1, 2011, [Online]. Available: http://cgit.openembedded.org/openembedded/plain/recipes/linux/linux-wrt-2.4.20/2.4.20_broadcom_3_37_2_1109_US.patch.
- [6] C. Chang, J. Wawrzynek, and R. W. Brodersen, "BEE2: a high-end reconfigurable computing system," *IEEE Design and Test of Computers*, vol. 22, no. 2, pp. 114–125, Feb. 2005, doi: 10.1109/MDT.2005.30.
- [7] "What is GNU?," *GNUs*, 2008. <https://www.gnu.org/home.en.html>.
- [8] Linux Kernel Organization, "The Linux Kernel Archives," *Kernel*, 2020. <https://www.kernel.org/>.
- [9] K. Yaghmour, *Building embedded Linux systems*, 1st ed. O'Reilly Japan, 2003.
- [10] P. Raghavan, A. Lad, and S. Neelakandan, *Embedded Linux System Design and Development*. Auerbach Publications, 2005.
- [11] A. A. Khan, "Practical Linux programming: device drivers, embedded systems, and the Internet," *Choice Reviews Online*, vol. 40, no. 03, pp. 40-1586-40-1586, 2002, doi: 10.5860/choice.40-1586.
- [12] "Busybox," *Busybox*. 2020, [Online]. Available: <https://busybox.net/>.
- [13] S. K. Kweon, M. G. Cho, and K. G. Shin, "Soft real-time communication over Ethernet with adaptive traffic smoothing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 10, pp. 946–959, Oct. 2004, doi: 10.1109/TPDS.2004.59.
- [14] K. C. Lee, S. Lee, and M. H. Lee, "Worst case communication delay of real-time industrial switched Ethernet with multiple levels," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 5, pp. 1669–1676, Oct. 2006, doi: 10.1109/TIE.2006.881986.
- [15] S. K. Kweon and K. G. Shin, "Statistical real-time communication over Ethernet," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 3, pp. 322–335, Mar. 2003, doi: 10.1109/TPDS.2003.1189588.
- [16] S. Wang, S. Malik, and R. A. Bergamaschi, "Modeling and integration of peripheral devices in embedded systems," in *Proceedings -Design, Automation and Test in Europe, DATE*, 2003, pp. 136–141, doi: 10.1109/DATE.2003.1253599.
- [17] J. M. De Goyeneche and E. A. Fernández De Sousa, "Loadable kernel modules," *IEEE Software*, vol. 16, no. 1, pp. 65–71, 1999, doi: 10.1109/52.744571.
- [18] K. J. Lin, S. H. Huang, and S. C. Fang, "Cooptimization of interface hardware and software for I/O controllers," in *Proceedings -*




- Design, Automation and Test in Europe, DATE*, 2006, vol. 1, pp. 1–2, doi: 10.1109/date.2006.244070.
- [19] M. Lewandowski, M. J. Stanovich, T. P. Baker, K. Gopalan, and A.-I. I. A. Wang, “Modeling device driver effects in real-time schedulability analysis: Study of a network driver,” in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, Apr. 2007, pp. 57–68, doi: 10.1109/RTAS.2007.18.
- [20] Y. T. Hsu, Y. J. Wen, and S. De Wang, “Embedded hardware/software design and cosimulation using user mode Linux and SystemC,” in *Proceedings of the International Conference on Parallel Processing Workshops*, Sep. 2007, pp. 17–17, doi: 10.1109/ICPPW.2007.39.
- [21] N. Cherukuri, G. B. Kandiraju, N. Gautam, and A. Sivasubramaniam, “Analytical model and performance analysis of a network interface card,” *International Journal of Modelling and Simulation*, vol. 24, no. 3, pp. 179–189, Jan. 2004, doi: 10.1080/02286203.2004.11442302.
- [22] A. Riadh Rebai and S. Hanafi, “An adaptive multimedia-oriented handoff scheme for IEEE 802.11 WLANs,” *International Journal of Wireless & Mobile Networks*, vol. 3, no. 1, pp. 151–170, 2011, doi: 10.5121/ijwmn.2011.3114.
- [23] Y. Guo and W. Deng, “Design of network device driver in embedded Linux,” in *ICCASM 2010 - 2010 International Conference on Computer Application and System Modeling, Proceedings*, Oct. 2010, vol. 12, pp. 445–448, doi: 10.1109/ICCASM.2010.5622349.
- [24] K. S. Parmar, S. Dessai, S. G. S. P. Yadav, and A. Chauhan, “Design and Implementation of an Ethernet MAC IP Core for Embedded Applications,” *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 3, no. 3, p. 85, 2014, doi: 10.11591/ijres.v3.i3.pp85-97.
- [25] S. H.G., S. Dessai, and S. Chaudhari, “Design of secure transmission of multimedia data using SRTP on Linux platform,” *International Journal of Reconfigurable and Embedded Systems (IJRES)*, vol. 4, no. 2, p. 71, 2015, doi: 10.11591/ijres.v4.i2.pp71-81.

BIOGRAPHIES OF AUTHORS



Prabhakar Pujeri    received his MSc Physics from Karnataka University Dharward and MEngg from Coventry University, UK, He is working in MNC, and his research areas are Embedded Systems, Operating Systems, Linux, and device drivers. He can be contacted at email: prabhakar.pujeri@gmail.com



Sanket S Naik Dessai    received his MSc Physics from Goa University, MEngg Real-Time Embedded System from Coventry University, MS Microelectronics from Manipal University, and Pursuing Ph.D., He had 16+ work experience in Industries and Academics. His research interests are Embedded Computer Architecture, System on Chip Design, Communication Systems, and their Signal Processing (includes SDR architecture development, SDR algorithms, SDR algorithms mapping for systems through SoC and multi-core architecture, 5G/4G/LTE systems, and algorithms, wireless and IoT Applications. He can be contacted at email: sanketdessai0808@gmail.com.