The Role of Embedded Linux in a Microwave Point-to-Point Radio Application

Wee Kiam Peng
Microwave Networks Department
Nera Telecommunications Ltd.
109 Defu Lane 10 Singapore 539225
kiampeng.wee@neratel.com.sg

Abstract

Embedding Linux in a carrier-grade application, like a Microwave Point-to-Point Radio Link, has many interesting concerns. These concerns range from technical issues, cost calculation issues to even political or licensing issues. However, when compared to an existing proprietry-only product which binds a company to long development cycles, it has many relatively obvious advantages. This report aims to address the concerns that have been raised in the process of embedding Linux in a microwave Point-to-Point radio application.

1 Introduction

Currently, point-to-point fixed microwave radios have been used in the telecommunications sector for transmission of data over areas impractical for laying cable. The cost of these radios are dropping [1] as market demand drives it. To be able to compete in the market where the time-to-market is being shortened, cost reduction methods are required to drive the cost down. As proprietry software has always been used for these type of radios, the cost/time of maintaining the changes and upgrades has been huge. One way to reduce product development costs is to employ open source software alternatives which is robust enough and easily customisable for the proprietry hardware.

Variants of Linux have been used in consumer routing equipment such as wireless lan routers [2] as well as wireless firewall devices for corporate use [3]. In anticipation of Linus Torvalds'es involvement with OSDL, Carrier-Grade Linux will soon become a reality when OSDL's Carrier-grade Linux group provides a standardised linux specification for carrier-grade applications [4]. This is currently in progress. These reasons combined will mean that using Linux for a carrier-grade product will be possible and will aid in reducing a product's time-to-market and per unit cost.

However, concerns do affect the route embedded Linux [5] take to these products. Traditional product development approaches tend to be biased against open-source methodology as opening up of software meant reducing the value of intellectual property. Previous investment in proprietry software components has to be justified and these usually lead to reusing software components that has been previously developed in-house.

This work-in-progress report describes a generic microwave point-to-point radio architecture and investigates how deviations to the normal process of software development for this architecture can occur. It also brings on the technical issues, cost issues and licensing issues to the table for further thought.

2 System Description

A microwave point-to-point radio link consists of two mirrored and similar nodes that are 'connected' through a radio channel (Air Interface). Figure 1 shows the basic functional units that constitutes the radio link. They are made of two symmetrical nodes.

The main difference between these two nodes is that the transmit frequencies of one node will be of the same value as the receive frequencies of the other, and vice versa. The **MU (Modem Unit)** and the **RFU (Radio Frequency Unit)** are self-contained within the radio link and the only external interfaces to the public are at the **IFU. (Interface Unit)**.

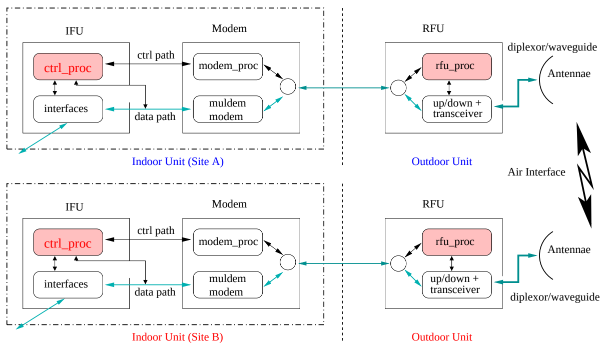

FIGURE 1: *A Simplified Microwave Radio Link*

In the traditional approach, where the classic superheterodyne receivers are used, the modem functions are usually located at the In-Door Unit while recently, there are approaches that bring the entire modem unit to the Out-Door Unit, in conjunction with advances in direct quadrature modulation techniques. [6]

For both approaches, the main controller software lies in the control unit which schedules most of the tasks that rank secondary in importance to the control of customer data channels.

The Interface Unit interfaces with the external environment and provides the radio link with a datapath input and output as well as control and monitoring signalling.

- Power Supply
- Service/Auxiliary Channels
- Telecommunication Standard Data Lines
- Network Management Ports
- Local Control Ports

Customer channel data, service/auxiliary channels are multiplexed, framed, coded, modulated, mixed and transmitted over the radio channel (air interface) while the receive chain reverses the process. This transmission/receiving chain represents the core data path of the radio link and has a fixed payload. Within this core data path that is sent over the air interface, customer channel data, auxiliary data as well as control information are embedded within this fixed payload.

While this is an oversimplification of the transmission/receive chain, it is adequate to illustrate the various conversions of digital data format that take place.

3 Hardware Architecture

The generic hardware architecture of the radio node consists of up to three individual processing elements (ctrl_proc, modem_proc (optional), rfu_proc) depending on the complexity of the radio as discussed previously. It may be possible to combine the modem controller functions with the secondary controller or the main controller to simplify designs. At each stage of the datapath, the processing elements provide specific functions such as being main controller (ctrl_proc) for the radio node as well as secondary controller (rfu_proc) for the outdoor radio unit.

An FPGA or a customised ASIC is normally used for providing data multiplexing, framing, modulation and error correction functions for the radio node. The main controller will therefore be used to control and configure the various states of the reconfigurable hardware (FPGA) or the customised ASIC.

Essentially, a microwave radio unit is a hardware product where the complexities lie mostly in the hardware domain. The embedded software will therefore be more of a tool to support the various states and functions that the hardware can provide.

4 Software Architecture

With the hardware architecture in mind, we will be able to understand how an embedded operating system can fulfil the role of a main controller by interfacing with the software components that sits above it. Before that, we will have to define the **Radio Functions** that are essential for the radio to operate properly. These functions are derived from the product specifications of the radio product.

4.1 Definition of Radio Functions

The chosen embedded operating system has to manage the system by providing functionalities that are related to the radio node as well as considering the implications and scenarios when a radio link is set up.

1. **Radio node/link config. and reporting** Allows raw configuration and reporting of radio node/link. (frequency, power, alarm, modulation, datapath settings)

2. **Traffic controller** Provides digital cross-connect setups or add/drop capabilities. selection of different customer channel data interfaces.

3. **Real-time OS capabilities** Decides on tasks that are critical and requires micro-management of real-time response and handling.

4. **Routing of network management traffic** Allows remote monitoring/control of node status/configuration which includes performance measurement details (ITU-T G.821, G.826 etc.) and configuration information.

5. **Configuration management** Tracks software and hardware production data as well as their revisions for compatibility checks as well as mechanism for software functionality upgrades.

6. **Diagnostic and loopback tests** Determines the up/down status of the hardware modules within the node. determines the up/down status of the datapath through the radio node and throughout the radio link.

7. **Auxiliary channel/interfaces usage** Provides control and configuration capabilities for auxiliary channels such as service channels, alarm relays etc.

4.2 Components Mapped to Radio Functions

With the abovementioned functions in mind, the software components are therefore arranged in the manner below (Fig. 2).

As we can see from the figure, embedded Linux has been used as the central management system (kernel) and it interfaces closely with the target microprocessor (ctrl_proc). On and above the kernel level are the functional components that are classified to three main component groups. They are the Interface Components, Network Management Components and the Control Software Components. These components are made up of proprietry (in red) and open-source modified (in blue) codes. The mapping of the functional components to the radio node functionalities listed previously are as follows

1. ctrl_sw, perf_mgt, bootloader, modcom, cit, boa

2. ctrl_sw

3. kernel

4. zebra, net-snmp, boa, perf mgt

5. misc_sw, boa, cit

6. ctrl_sw

7. pppd, voip, ctrl_sw, ifcom

This is a simplified mapping that is carried out in the design requirements stage of the development cycle. ctrl_sw is a user-mode application which provides most of the background control of various subsystems in the radio node. It will require real-time-ready interfaces with the other components such as the modcom (modem communication), ifcom (interface communication) as well as the network management components.

Zebra, net-snmp and boa are well-tested open source applications that provide dynamic routing, network status reporting as well as web interface functionalities respectively. They require minimal changes to fit to the system.

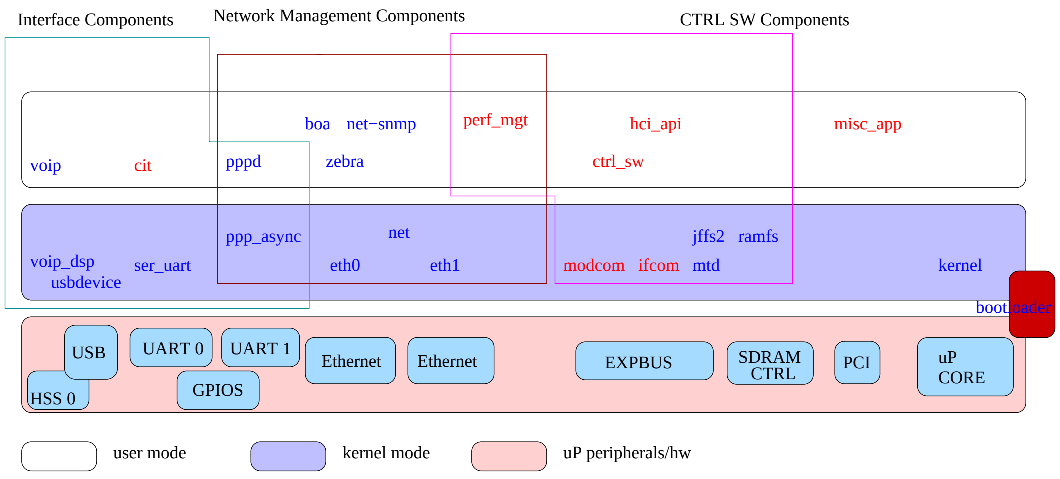

FIGURE 2: *Software Architecture*

4.3 Availability of Components

From the previous diagram (Figure 2), we can estimate that almost fifty percent (or even more) of the software components is already available. With slight modifications, they will be able to be customised to requirements. This implies that for a product development cycle, previous development time can be utilised for focusing on critical components such as modcom, ifcom as well as the ctrl_sw or that the software development time can be reduced. The following table also illustrates the complexities of the software versus the availability before the start of the software development cycle.

Table 1 Availability/Complexity of Components

Component	AVAILABILITY	COMPLEXITY
kernel	yes	highest
zebra	yes	high
ctrl_sw	no	high
net-snmp	yes	high
boa	yes	normal
perf_mgt	no	normal
pppd	yes	normal
cit	no	normal

TABLE 1: *Availability/Complexity of Components*

4.4 Other Issues

From a technical and product development point of view, it seems like the usage of embedded Linux will be an advantage over totally proprietry software systems for this application.

However, there are other issues to be aware of. One of the pressing questions that management boards ask is whether there will be legal issues involved in using embedded Linux. As this is still an evolving matter with Linux having acquired strong legal ground, we see that the future is bright.

The second issue that might be worth addressing is due to a strong proprietry-only development environment. It will be difficult for these sort of users to justify a switch to embedded Linux as costs have already been incurred in running development licences and development work that was already done.

5 Conclusion

This short report addresses the concerns that has come about when using embedded Linux in a microwave point-to-point radio application. It describes the hardware-centric nature of these types of products. At the same time, it tries to illustrate how a pre-development evaluation is being carried out by grouping the software components according to availability and complexity and then eventually map them to the radio node's subsystems acording to defined radio functions that are extracted from product design specifications.

Through the exercise where radio functions are defined for software components, it has shown that it might be possible to cut development costs and also reduce product development time when compared to a development cycle that favours proprietry-only concepts. However, this is on assumption that the proprietry-only project requires major coding.

The role of embedded Linux for a microwave point-to-point radio application is to aim to provide a low setup cost, consistent, licence-free, extensible and robust environment to satisfy the defined radio functional requirements.

References

[1] Q4 2003, *Microwave Forecast Report*, Skylight Research.

[2] http://www.seattlewireless.net/ index.cgi/LinksysWrt54g

[3] http://www.cyberguard.com/ snapgear/products.html

[4] http://www.osdl.org/

[5] G. Ungerer, *uClinux, Micro-Controller Linux*

[6] A. Abidi, Dec. 1995, *Direct-conversion radio tranceivers for digital communications*, IEEE J. Solid-State Circuits, vol 30, pp. 1399-1410, ISBN.

[7] D.P. Bovet,M. Cesati, 2003, *Understanding the Linux Kernel*, O'reilly.