ENGN3214 - Telecommunications Systems

Hugh Blemings

A project report forming part of the course requirements of ENGN3214 in Semester 1, 2006.
Faculty of Engineering and Information Technology,
Australian National University

June 2006

Except where otherwise indicated, this document is my own original work. Any trademarks and product names that appear in this document are the property of their respective owners.

© Hugh Blemings, 2 June 2006

Typeset in Palatino by TeX and LaTeX 2_ε.

Acknowledgements

While personal thanks are unconventional in a short technical work such as this, I none the less thank my wife Lucy and daughter Rachael for their support throughout my studies.

I took ENGN3214 as an elective toward completing the ANU's Masters of Information Technology / eScience course part time. As such it was my first foray into the Engineering school after a diet solely comprised of Computer Science courses. I thank Dr Gerard Borg and my fellow students for their part in making this such a pleasant and engaging experience.

The support of my employer, IBM and my colleagues and management in the Linux Technology Centre is very much appreciated and I extend my thanks to them also.

Hugh Blemings, 2 June 2006

Contents

Acknowledgements **iii**

1 Introduction **1**
- 1.1 Basis of the project . . . 1
- 1.2 Overview of this report . . . 2

2 Design Considerations **3**
- 2.1 Introduction . . . 3
- 2.2 Automatic Frequency Control . . . 3
- 2.3 Clock and Data Recovery . . . 3
  - 2.3.1 Common Design Issues . . . 4
- 2.4 Modulator / DAC . . . 4
- 2.5 On air protocol / MAC Layer . . . 4

3 Phase Locked Loops **5**
- 3.1 Introduction . . . 5
- 3.2 Lab Work - Clock Recovery PLL . . . 6
  - 3.2.1 Some Results . . . 7

4 System Hardware **9**
- 4.1 Introduction . . . 9
- 4.2 To Rabbit, or not to Rabbit . . . 9
  - 4.2.1 Background . . . 9
  - 4.2.2 Some Gotchas . . . 10
- 4.3 Requirements of alternative hardware . . . 11
- 4.4 D-Link NSLU-2 & OpenSLUG . . . 11
- 4.5 ASUS WL-500G Deluxe . . . 12
  - 4.5.1 Hardware Description & Modifications . . . 13
- 4.6 A cheap, fast DAC and ADC solution . . . 13
  - 4.6.1 Suggested Hardware Modifications . . . 15
- 4.7 Disadvantages to re-purposed hardware . . . 15
- 4.8 Linksys WRT-54 series . . . 16

5 System Software **17**
- 5.1 Introduction . . . 17
- 5.2 (Affordable!) Embedded Linux . . . 17
- 5.3 Soundmodem and AX25 support under Linux . . . 18

5.3.1 Kernel support . . . 18
5.3.2 AX25 User Space Utilities and Libraries . . . 18
5.3.3 Soundmodem software . . . 18
5.4 OpenWRT . . . 19
5.4.1 Basic installation . . . 19
5.4.2 Customisation . . . 21
5.4.3 Customisation Results . . . 22
5.4.4 Building Hello World! . . . 23
5.4.5 Concluding Remarks . . . 23

6 Conclusions **25**
6.1 Next Steps . . . 25
6.2 Future work . . . 26

A Appendix **27**
A.1 Other Sources . . . 27

Bibliography **29**

Chapter 1

Introduction

1.1 Basis of the project

This work is based on the project requirements set out for the ENGN3214 and ENGN4521 classes in Semester 1, 2006. Quoting from the project documentation [Borg 2006] ;

> The project part of the ENGN3214 and ENGN4521 courses covers the design and construction of a complete radiofrequency (Very High Frequency (VHF) 27 - 41MHz) BPSK/FSK radio. Execution of this project in a timely fashion represents a significant if not impossible undertaking.
>
> After this project you will, at least hopefully, appreciate an underlying complexity that will not be evident a-priori on the basis of the material covered in either course. The data radio is for communication of information over TCP/IP networks at low bit rates (10 kbps) and over serial telemetry links. The aim by the end of the course is to build a number (3-4) of these devices. A pair of these devices can be connected each to a tuned whip antenna and plugged into separate LANs at each end of a wireless link. The system then behaves as a "pipe" in the sense that the two LANs are transparently bridged across the wireless link.

As originally defined, the work expected of the ENGN3214 was essentially confined to the baseband processing part of the project namely;

- Carrier Recovery
- Data and Clock Recovery
- Controller/MCU programming
- Modulator/DAC

It was expected that a mixture of analysis and modelling, prototyping and measurement and software development would be completed. As the semester progressed some areas received a greater emphasis, others less.

Early in the course volunteers were sought to undertake the development of the microcontroller side of the project including any 'C' coding that might be required.

After some discussion it was agreed that the author would undertake this aspect of the project and provide his work back to the rest of the class. Having a single code base on the controller side would also avoid any compatibility issues between implementations.

Accordingly this report has something of a different emphasis focussing for the most part on the research into the controller/MCU side of the project.

1.2 Overview of this report

The remainder of this report contains;

- General discussion about design considerations.
- Notes on Phase Locked Loops.
- Control system hardware & software.
- Conclusions and future work

Chapter 2

Design Considerations

2.1 Introduction

In this chapter we will discuss some of the general design considerations and issues that arise when designing base band processors such as the one at hand.

To begin we present some background on Automatic Frequency Control.

2.2 Automatic Frequency Control

Automatic Frequency Control (AFC) is a mechanism that allows the frequency of a local oscillator or clock reference to be locked to an incoming or otherwise external signal.

For transmission systems where a carrier is present this clock recovery is typically fairly trivial. It is often sufficient to retrieve the carrier, clean it up, possibly by use of a PLL, and your work is pretty much done.

However, many popular transmission techniques deliberately reduce or remove the carrier in order to allow as much energy as possible to be dedicated to the information signal. In these situations a more complex system is required to account for the absence or changing nature of the carrier.

2.3 Clock and Data Recovery

Clock and Data Recovery is the process of retrieving clock and data information from an incoming baseband signal. The techniques required for each are usually related hence they will be discussed together.

For many modulation/demodulation techniques, successful recovery of a data stream from a baseband signal will rely on close if not exact synchronisation between the transmitter and receivers clock signal(s). This was perhaps most vividly demonstrated in our course when we looked at QAM signals where it was all but impossible to see the true constellation unless the LO and incoming signal were within a few tenths of a hertz.

In lectures and labs we examined a number of PLL based CDR designs, Costas Loop, Squaring Loop and the Cross Product detector among them.

2.3.1 Common Design Issues

Each CDR design had it's strengths and weaknesses, these are not treated in this report in any depth.

One common issue however is to consider how the operation of the clock recovery circuit places requirements on the transmitted datastream. Many modulation schemes used in contemporary systems do not provide a separately transmitted clock, instead relying on the receiver to recover this from the data stream. This in turn places requirements on the data stream, for example as it goes to air it can never be all ones or all zeroes (or otherwise in a steady state) as this will leave nothing for the receiver to track.

2.4 Modulator / DAC

The planned modulation scheme for our system was BPSK. With BPSK systems the carrier phase is either left unchanged (logic '0') or shifted $180°(logic'1')$.

In its most crude form, this can be achieved by a multiplier in the carrier circuit that multiplies the LO by 1 or -1 depending on the data. This simplistic approach is far from optimal as it does nothing to achieve smooth transitions and reduce ISI at the receiver. Accordingly most practical systems will use a DAC or pulse shaping circuit so the resultant signal is shaped to some degree.

2.5 On air protocol / MAC Layer

The project outline provided made no particular recommendations about the on air protocols to be used. After discussions with the lecturer it was agreed that the AX25 protocol [Beech et al. 1998] would be an appropriate choice.

AX25 was developed by the Amateur (or Ham) radio community in the early eighties for use as a packet radio data network. Its design is optimised for imperfect links operating in what would now be called a wide area network. The design of the medium access control parts of the standard provide reasonable performance even in tricky radio environments such as can occur when not all stations are able to directly receive each others transmissions (aka the "Hidden Transmitter Problem").

Although original designed with keyboard to keyboard and BBS style usage in mind, the encapsulation of TCP/IP over AX25 is standardised and will be used in our project.

As will be discussed more fully in Chapter 5 the choice of AX25 had the further advantage of allowing us to leverage existing Open Source software to provide the necessary operating system and modem support.

Chapter 3

Phase Locked Loops

3.1 Introduction

A Phase Locked Loop (PLL) is a circuit block that provides an output signal that is synchronised in frequency and phase with the input signal of interest. A PLL consists of three basic components - a phase detector, a low pass filter and a voltage-controlled oscillator (VCO)

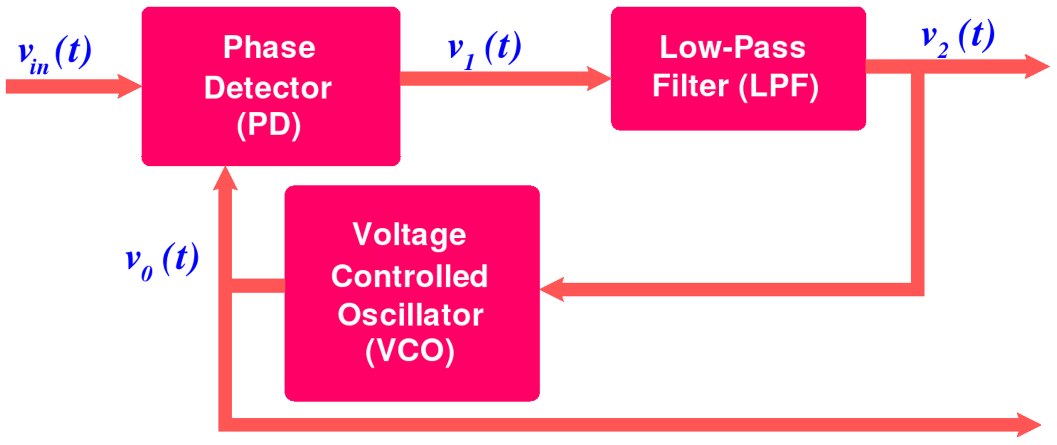


Figure 3.1: PLL Block Diagram (from [Borg 2006])

The phase detector provides an output voltage proportional to the difference in phase (and hence potentially frequency) between the two input signals.

The low pass filter is carefully chosen to provide a response such that the control voltage it feeds to the VCO will bring the VCO's output into phase with the incoming signal with minimal undershoot/overshoot.

The VCO outputs a periodic waveform thats frequency is determined by the input voltage. It will typically have a centre or free-running frequency, in most applications this will be set to be in the mid-range of the expected input frequencies the PLL is to track.

PLLs have a number of applications including [Borg 2006]

- Frequency Multiplication
- Production of Frequency Modulated signals
- Frequency Synthesis
- Clock recovery in digital communications
- Coherent AM detection

In the project we focussed on the clock recovery and frequency synthesis (for the LO)

As an aside, while PLLs are most often thought of in analog or RF applications, most modern microprocessors make extensive use of them to generate internal clock(s) referenced to a single, external, low frequency crystal.

3.2 Lab Work - Clock Recovery PLL

In labs the author undertook the design of a clock recovery PLL circuit suitable for the system at hand. The 74HC4046 integrated PLL chip had been chosen and so reference was first made Fairchild's data sheet [Fairchild Semiconductor 1999]. When it was found to be overly opaque, one produced by Phillips was used instead. [Phillips Semiconductors 1997]

In review of the lecture notes and in lab discussions with the lecturer and classmates it seemed that the following basic criteria would be appropriate given our system bit data rate of 10kbps.

- Centre frequency of 10kHz
- Minimum frequency of 7.5kHz
- Maximum frequency of 15kHz

The frequency of operation of the VCO in the '4046 is set by two resistors (R1 & R2) and a capacitor (C1). Working through the Phillips datasheet using the parameters above suggested values of R1 $\simeq$ 5k9, R2 $\simeq$ 9k9 and C1 $\simeq$ 39nF.

A circuit was breadboarded up using closest available values to the above - R1 = 5k9 (4k7 + 1k2), R2 = 10k and C1 = 44nF (22nF + 22nF)). With the VCO control voltage set to VCC/2 the output (centre) frequency was measured at 9.9kHz.

Using an adjustable power supply for the VCO control voltage suggested a range of 6.9kHz to 14.8kHz, adequate for our purposes.

Referring to the datasheet and conferring with the lecturer suggested that the 74HC4046's Phase Comparator 2 circuit would be most suitable. In particular it claimed to have good immunity to second harmonics of the frequency of interest.

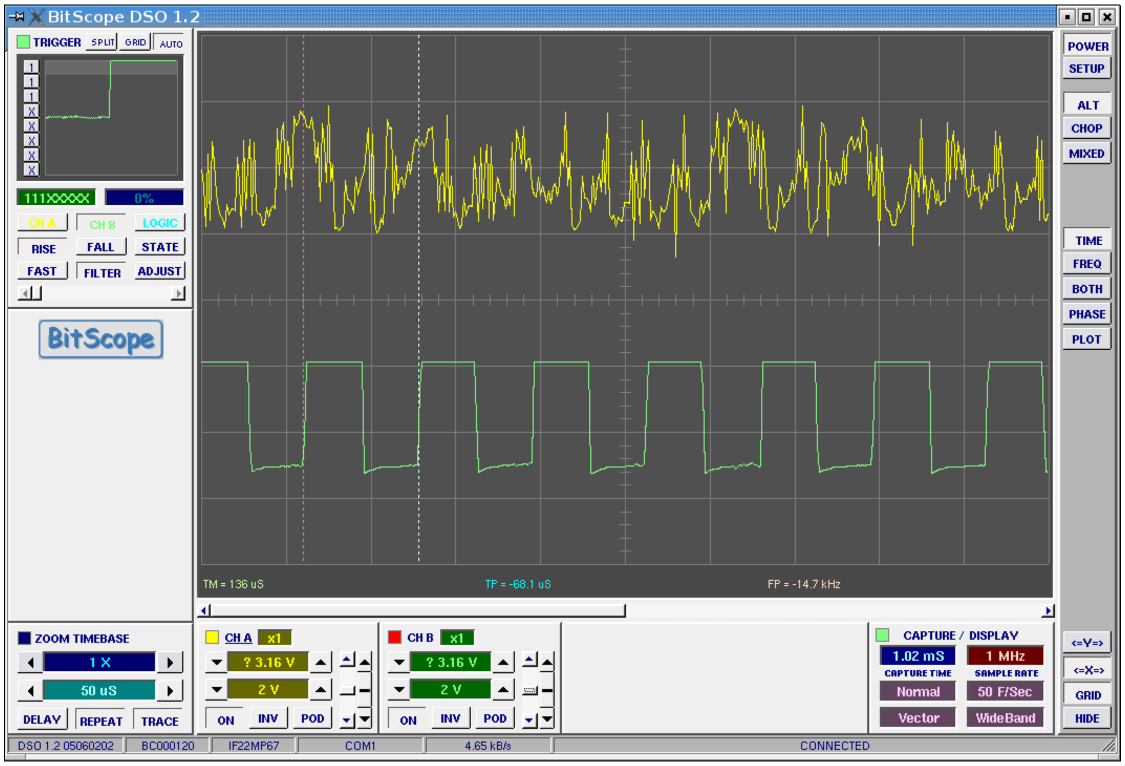
As a first experiment, a simple RC loop filter was designed using the information provided in the Phillips datasheet. This suggested values of R3 $\simeq$ 11k8, R4 $\simeq$ 370R and C2 $\simeq$ 470nF.

This circuitry was prototyped with R3 = 12k, R4 = 390R and C2 = 470nF and the input driven from one of the signal generators in the lab. The circuit was found to lock nicely to incoming frequencies in the range 7.5kHz to 12kHz. Empirically frequencies at harmonics of the PLL frequency did not cause it to gain lock.

Unfortunately when fed with a sweeped signal it was found that the PLL would only accurately track when the frequency was ramped up. On the downward frequency sweeps the PLL would at worse lose lock or at least not track very cleanly, something further evidenced by monitoring the control voltage. The author did not have time to investigate this further but would speculate that the loop filter was either incorrectly designed and/or too simple to operate effectively.

3.2.1 Some Results

The author did not have the opportunity to capture data from the circuit in the lab due to time constraints, however two captures were done at home just prior to report submission and are included in Figures 3.2 and 3.3.



TM = 136 uS   TP = -68.1 uS   FP = -14.7 kHz

ZOOM TIMEBASE: 1 X, 50 uS

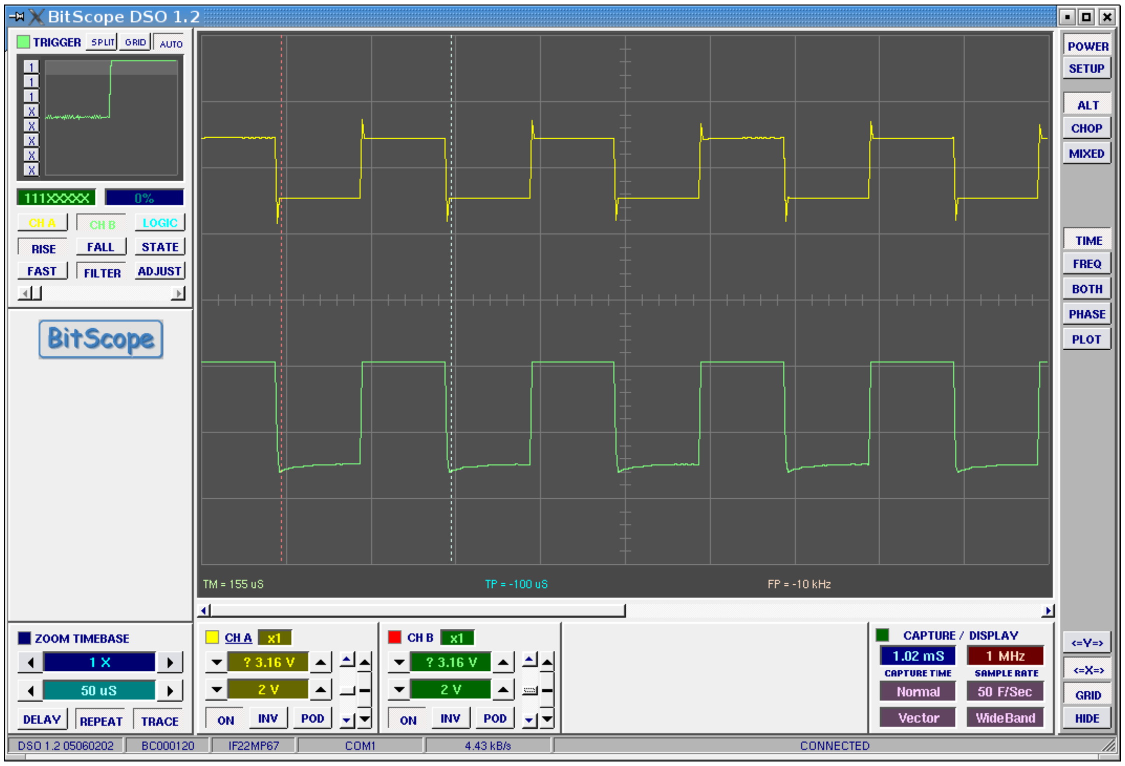
CH A x1: ? 3.16 V, 2 V

CH B x1: ? 3.16 V, 2 V

CAPTURE TIME: 1.02 mS, Normal, Vector — SAMPLE RATE: 1 MHz, 50 F/Sec, WideBand

DSO 1.2 05060202 | BC000120 | IF22MP67 | COM1 | 4.65 kB/s | CONNECTED

Figure 3.2: PLL output (lower trace) when unlocked, noise on input (upper trace)



111XXXXX 0%

TM = 155 uS TP = -100 uS FP = -10 kHz

ZOOM TIMEBASE 1 X 50 uS

CH A x1 ? 3.16 V 2 V

CH B x1 ? 3.16 V 2 V

CAPTURE / DISPLAY 1.02 mS CAPTURE TIME 1 MHz SAMPLE RATE 50 F/Sec

DSO 1.2 05060202 BC000120 IF22MP67 COM1 4.43 kB/s CONNECTED

Figure 3.3: PLL output (lower trace) when locked to incoming 10kHz square wave (upper trace)

Chapter 4

System Hardware

4.1 Introduction

In the following two chapters, we examine the controller side of the project. This chapter covers hardware, software is dealt with in Chapter 5.

As noted previously, the author put the majority of his effort into this facet of the project having volunteered to provide an implementation that could be shared among the whole class.

The project required the microprocessor and associated software to perform a number of tasks;

- Generate signals for data modulation
- Decode the incoming data and clock signals (demodulation)
- Monitor the radio RSSI signal to determine channel availability
- Control TRX switching
- (Potentially) Control the frequency synthesiser in the RF subsystem
- Implement a suitable on-air protocol
- Communicate with the attached ethernet network and serial devices

We will now discuss the choice of hardware for the system.

4.2 To Rabbit, or not to Rabbit

4.2.1 Background

The original project specification recommended the use of a Rabbit Semiconductor microcontroller board to perform the required computational duties.

Based around the Rabbit 2000 microprocessor (a Z80 derivative) the RCM2100 module supports some 34 CMOS compatible parallel I/O lines, two asynchronous UARTs, various timers and, importantly for this application, 10-Base-T Ethernet. It also has a useful amount of Flash and SRAM storage (256kB and 512kB respectively). [Rabbit Semiconductor 2000]

Rabbit make available a royalty free TCP/IP stack and Ethernet drivers for the board as part of their SDK which makes it fairly easy to get basic socket based applications going.

4.2.2 Some Gotchas

In digging in to what would be required of the board for the project it became clear that there were a few of potential stumbling blocks.

- The supplied TCP/IP stack and Ethernet drivers are designed with client/server style applications in mind. A socket library was provided (including the ability to listen on multiple ports) however there was no obvious way to trivially implement any form of routing without writing a lot of code ourselves.

- The networking code relied on being able to make use of a reasonable amount of the available processor cycles. The documentation seemed to intimate that this could impact interrupt latency for other code paths.

- While the target data rate was not high (10kbits/s) the author had some concerns about being able to service interrupts on the receive side and achieve sufficiently deterministic timing to generate accurate transmit bit streams for.

- The RCM2100 boards are relatively cheap even in low quantities - around AUD$120 each. However the overall bill of materials for the processor side of the finished system would add up to several hundred dollars by the time an ethernet switch and other necessary devices were added.

- The C compiler supplied with the board SDK used a Rabbit specific set of proprietary extensions to support the board. It also required some curious fiddling around to achieve what would otherwise normally be done with conventional `#include` directives and object file linking.

- Despite many hours experimentation with WINE, vmware and qemu, there appeared to be no way to get the full tool suite to work properly under Linux. [1]

In considering the above, making use of the Rabbit embedded board began to look rather less attractive so an alternative was conceived - re-purpose suitable existing hardware and leverage Open Source software to provide the projects computational core.

[1] Ironically once installed on a vanilla Windows XP machine the serial debugging link remained somewhat flaky.

4.3 Requirements of alternative hardware

To be suitable for the project the re-purposed target device would ideally need to provide the following;

- One or two fast ($\geq 20kHz$), low CPU overhead output ports or DAC channels to generate transmit waveforms. Two channels would provide the hardware required to allow IQ modulation techniques should this be desired.
- One or two fast ($\geq 20kHz$), low CPU overhead input ports or ADC channels to accept the baseband receive signals (data and clock signals)
- Lower speed I/O ports to monitor RSSI and provide TRX switching
- Ethernet and serial port(s)

4.4 D-Link NSLU-2 & OpenSLUG

Figure 4.1: DLink NSLU-2 (Image ©DLink, Inc. 2004)

The D-Link NSLU-2 (Figure 4.1) is a Network Storage Appliance designed to allow up to two external USB hard drives to be shared over a local 100-Base-T ethernet network. It makes use of a ARM based System on Chip (SoC) processor and has 32MB of RAM and 16MB Flash.

By modifying the factory hardware with the addition of a line transceiver, it is possible to add one async serial port with RXD and TXD support and a second async port with RXD only. There are three general purpose I/O pins left available that can be accessed with judicious use of a soldering iron.

The NSLU-2 is well supported by the OpenSlug project [2] which provides an alternative firmware stack and a cross compilation environment so that people can build their own custom firmware images.

[2]http://http://www.nslu2-linux.org

The NSLU-2/OpenSLUG combination looked to be a strong contender however it had two drawbacks. Firstly it lacked multiple ethernet ports or 802.11a/g wireless support which would be useful for the final application of the radio system. Secondly, in the authors experience OpenSLUG was relatively harder to customise for the task at hand.

Fortunately a better option was found in the ASUS WL-500G Deluxe which we now examine in detail.

4.5 ASUS WL-500G Deluxe

Figure 4.2: ASUS WL-500G Deluxe (Image ©ASUS TeK Computer Inc. 2004)

The ASUS WL-500G Deluxe [3] (Figure 4.2) is a combined network storage appliance and wireless router/gateway. Unmodified it provides two USB 2.0 ports, an 802.11a/g wireless LAN interface, a 100-Base-T WAN port and a four port, 100-Base-T LAN switch which is internally connected to a second 100-Base-T ethernet interface. The CPU is a Broadcom 5365 SoC (Mips architecture) running at 200MHz with 4MB flash and 32MB of RAM standard.[4]

With the addition of line transceivers it can provide two RXD/TXD asynchronous serial ports - headers are provide for simple hookup. There are a couple of GPIO pins left available too.

These units are readily ordered locally at a cost of around AUD$170. This coupled with good software support and decent baseline I/O capability led to this unit being chosen for the project.

The OpenWRT software stack, a key part of making this approach viable, is covered in the next chapter.

[3]The "Deluxe" designator is significant - the other WL-500G models lack a UART

[4]More details at http://wiki.openwrt.org/OpenWrtDocs/Hardware/Asus/WL500GD

4.5.1 Hardware Description & Modifications

The WL-500G is a single PCB internally with all rear panel connections being directly soldered to the PCB. The standard power supply is rated at 5V@2A, this appears to be further regulated internally down to a 3V3 supply for most of the onboard logic. There are four conveniently positioned mounting holes at the edges of the PCB and the plastic rear panel slides out of the case meaning it can either be used as a template to fabricate a new back panel or perhaps be incorporated as is.

In order to neatly integrate the WL-500G into the finished system, the author would suggest it be removed from its case and mounted toward the rear of the new enclosure so that the ethernet and wireless ports can still be reached. Status leds can trivially be extended to the front panel with hookup wire.

The 5V supply could be provided by the overall system supply so the input plug could be removed from the back panel.

One of the USB ports will be used to connect to the DAC/ADC device (see 4.6) - it is suggested that this be wired internally and the port that is so used be removed or blanked out so it is not available from the rear panel.

Alternatively, discussion on the OpenWRT forums about the WL-500G indicates that a third and fourth USB port is available on the PCB[5] which could be used instead. This *may* be contrary to the USB specification - the author recollects that the USB power supply should be switched and current limited. In practice it is unlikely that this will matter - something the empirical evidence on the forum seems to support!

If asynchronous serial ports are required, the addition of a MAX2323 style line transceiver will allow two RXD/TXD port pairs to be made available. The OpenWRT Wiki provides details of this modification.

4.6 A cheap, fast DAC and ADC solution

To provide fast and inexpensive ADC and DAC for our re-purposed router it was decided to investigate USB audio interfaces. The explosion in VoIP usage has seen a number of cheap (sub AUD$20) devices appear providing stereo audio out and a mono microphone input for use with inexpensive headset/microphone combinations. Figure 4.3 shows two typical examples, the top unit was bought over eBay, the lower one locally through a computer fair.

Between disassembling the units, judicious use of a 'scope and examining the USB vendor and product IDs the author determined that both made use of the C-Media CM108 USB Audio controller chip.[C-Media Electronics Inc. 2004] This chip suits the project well as it provides;

- Two 16 Bit 44.1/48kHz DAC (Line/Speaker)
- One 16 Bit 44.1/48kHz ADC (Microphone)

[5]http://wl500g.info/showthread.php?t=1779

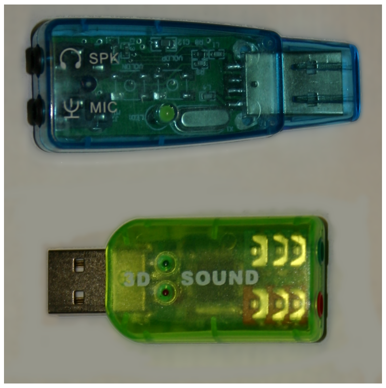


Figure 4.3: USB Audio interfaces

- Four programmable GPIO pins
- Three button inputs
- Provision to directly interface with an external D/A or A/D converter

The two DAC channels are suitable for providing IQ modulation signals to the transmitter. Closer examination revealed that the common signal on the output socket was biased at 2.5V - this presumably to allow an otherwise single ended output stage to provide +/-ve voltage swings. The downside of course being that if fed into a ground referenced circuit things would get warm.[6] A better option would be to use the signals referenced to ground and convert to bipolar if required in subsequent stages.

The single microphone input/ADC channel could be used as a input direct from the baseband circuitry leaving the clock and data recovery to be done in software. It is unclear however if sufficient processor bandwidth would be available to do full software based demodulation.

An alternative approach would be to make use of the CM108's ability to interface with an external A/D converter. This interface expects synchronous serial data clocked in at 44.1 or 48kHz as shown in Figure 4.4.

Simple external logic would allow the DATA and CLK signals from our recovery circuits to be alternately multiplexed in as "left" and "right" signals into the SDATA pin. The resulting word data from the kernel would have '1's when the pins were high, '0's when low. This would allow far simpler decode processing as the receive routine would merely need to look for a 0 to 1 transition on the CLK channel and sample the DATA channel on the next sample to recover the receive bit train.

The four GPIO pins provided on the CM108 are readily accessible via the USB HID class. These pins would allow monitoring of the RSSI signal, provision of TRX switching with a couple of pins spare for future expansion.

[6]Indeed it was the smell of warm electronics that first drew this to the authors attention...

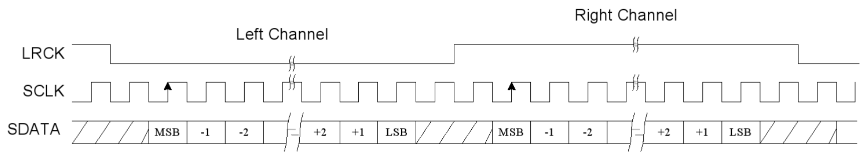


Figure 4.4: CM108 I2S Interface (From [C-Media Electronics Inc. 2004])

4.6.1 Suggested Hardware Modifications

The USB connector on the board can be removed and a short length of USB cable used to directly wire it to the USB port on the router. This will allow the interface to sit within the system enclosure.

As noted earlier, the DAC outputs on the headphone socket are referenced to 2.5V. A more elegant approach would see the relevant signals tapped off the PCB at a convenient point then fed into a small dual rail op-amp circuit to produce a true bipolar signal of appropriate amplitude.

To make use of the GPIO pins on the CM108 it will be necessary to solder onto the chip directly as they are not brought out to pads on the PCB. This will be fiddly as the pins are on 0.5mm centres, but not impossible.

One of the two USB Audio interfaces had provision for the "Volume Up" and "Volume Down" making at least these two inputs readily accessible.

4.7 Disadvantages to re-purposed hardware

It is worth noting a few potential disadvantages of the re-purposed hardware approach.

Most obviously, modifying the hardware will void any warranties from the manufacturer. It is therefore worth investing a little time to fully check the units out prior to modification. This may be nothing more than allowing it to run on the bench for 24 hours and checking that all the USB, ethernet and wireless interfaces function as expected.

These are consumer devices and so the exact design may change over time to suit the requirements of the manufacturer. Typically these sorts of changes are rapidly identified in communities like OpenWRT. It may be that the manufacturer model number doesn't change yet the underlying hardware is quite different, again this is usually noted on websites like OpenWRT. Most devices have some sort of serial number or batch code that can be referenced back to the information on the web.

Worst case this may mean looking for an entirely new device to re-purpose. It is felt that the inconvenience this may temporarily cause is far outweighed by the convenience an inexpensive, Linux based solution brings in terms of portability.

4.8 Linksys WRT-54 series

It would be remiss to end a discussion on re-purposed hardware without mentioning the Linksys WRT-54 series of routers. These were both the first devices identified "in the wild" running Linux that could be modified and the ones that gave the OpenWRT project the "WRT" in its name.

To their credit Linksys recognised the work of the community and when the -54G model was recently changed over to a proprietary embedded OS, Linksys released a parallel model (WRT-54GL) which still ran Linux albeit at a few dollars more.

The WRT series make an excellent, affordable Linux based router, but lacking USB do not suit the requirements of this project.

Chapter 5

System Software

5.1 Introduction

The previous chapter dealt with the hardware aspects of the project's computing requirements. This chapter will focus on the software side of things.

5.2 (Affordable!) Embedded Linux

Linux as a server or desktop operating system needs no introduction. What is somewhat less obvious is the extent to which Linux is being used as the base operating system for dozens if not hundreds of embedded devices. A great proportion of these are consumer applications where the underlying OS is not visible to the end user.

The most common applications for embedded Linux fall into the broad categories of network devices and storage appliances. The former are typified by wireless access points, ADSL modems and ethernet routers/gateways. The latter are a smaller group, mostly consisting of devices that allow a number of IDE or USB based hard disks to be shared over a (wired) ethernet LAN.

Processor wise the market is dominated by ARM and MIPS based cores with a smattering of PowerPC. Typical clock frequencies run into the 200MHz range with 16-32MB of RAM and 4-8MB Flash being the normal design points. We discussed two examples of this hardware in Chapter 4, sections 4.4 and 4.5.

For the most part manufacturers of these devices comply with the GPL license that covers the Linux kernel and make the relevant source code available[1]. This source code is used as a basis for various alternative, open source, firmware projects of which more below.

Before reviewing OpenWRT, our choice of Linux system for the project, we describe the application specific software that will be required.

[1]You can read about the exceptions at http://www.gpl-violations.org

5.3 Soundmodem and AX25 support under Linux

The Linux kernel has provided support for ham radio devices including AX25 networking and radio modem support for many years.

Beyond the defaults installed on most Linux systems, our application will require four pieces of additional software.

1. Linux Kernel AX25 support
2. Linux Kernel USB Audio support
3. AX25 user space utilities & libraries
4. Soundmodem software

5.3.1 Kernel support

The addition of the required kernel support is straightforward - when building a kernel and/or modules for our system we will need to ensure that AX25 and USB Audio support is included.

This will be achieved through the usual kernel configuration process.

5.3.2 AX25 User Space Utilities and Libraries

A suite of applications has been written to provide comprehensive support for the kernel's AX25 capabilities. Of interest for our project will be the ability to manage routing of TCP/IP traffic over AX25 links.

This application suite and library will need to be built for the target system.

5.3.3 Soundmodem software

Soundmodem is a pair of applications written by Thomas Sailer[2] that implement a variety of modems in software which interface with the kernel's AX25 and sound infrastructure.

`soundmodem-config` is a GUI based combined configuration and diagnostic tool. It allows the user to configure various software modems and their associated parameters. Visual displays of an oscilloscope, spectrum analyser and dump of the modem output are also available.

`soundmodem` is the soundmodem daemon proper. Run from the command line it picks up the configuration file generated by `soundmodem-config` and plugs in to the kernels sound and networking layers to implement relevant devices.

[2]http://www.baycom.org/ tom/ham/soundmodem/

5.4 OpenWRT

The OpenWRT project (http://www.openwrt.org) is an Open Source project that provides fully customisable firmware for Mips based devices such as the Linksys WRT-54G and ASUS WL-500G Deluxe that we are using.

OpenWRT and it's customisation will now be discussed in detail.

5.4.1 Basic installation

The OpenWRT website allows you to either download pre-compiled firmware images or pull down an entire environment to roll your own. The author went for the latter option and did a trial run building an unmodified version of the "WhiteRussian" release. The download and build process is well documented on the OpenWRT Wiki (http://wiki.openwrt.org)

The complete build took of the order of 60 minutes the first time, at least some of this being spent downloading additional packages in source code form. Subsequent builds were rather shorter - more of the order of ten minutes depending on what was being rebuilt[3]. Some 1.2GB of disk was needed to do a complete build including a cross compilation environment.

Uploading the resultant firmware image to the WL-500G was straightforward, making use of the instructions on the OpenWRT website.[4]

The results were most gratifying:

```
hugh@portnoy:~  ssh root@192.168.73.215
root@192.168.73.215's password:


BusyBox v1.00 (2006.05.11-03:00+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 WHITE RUSSIAN (RC5) -------------------------------
  * 2 oz Vodka   Mix the Vodka and Kahlua together
  * 1 oz Kahlua  over ice, then float the cream or
  * 1/2oz cream  milk on the top.
 ---------------------------------------------------
root@OpenWrt:~# uname -a
Linux OpenWrt 2.4.30 #1 Thu May 11 12:55:16 EST 2006 mips unknown
```

[3]Timings are based on the authors 2GHz AMD64 system "portnoy"

[4]http://wiki.openwrt.org/OpenWrtDocs/Hardware/Asus/WL500GD

Once logged in it's easy to dig around and get various bits of information about the system.

```
root@OpenWrt:~# cat /proc/cpuinfo
system type             : Broadcom BCM947XX
processor               : 0
cpu model               : BCM3302 V0.7
BogoMIPS                : 199.47
wait instruction        : no
microsecond timers      : yes
tlb_entries             : 32
extra interrupt vector  : no
hardware watchpoint     : no
VCED exceptions         : not available
VCEI exceptions         : not available
```

It is worth noting that the CPU does not have a hardware FPU - a potential issue if floating point intensive signal processing code is contemplated.

```
root@OpenWrt:~# cat /proc/meminfo
        total:    used:    free:  shared: buffers:  cached:
Mem:  31289344  8069120 23220224        0        0  2945024
Swap:        0        0        0
MemTotal:        30556 kB
MemFree:         22676 kB
MemShared:           0 kB
Buffers:             0 kB
Cached:           2876 kB
SwapCached:          0 kB
Active:           1388 kB
Inactive:         1504 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:        30556 kB
LowFree:         22676 kB
SwapTotal:           0 kB
SwapFree:            0 kB
```

The system makes very efficient usage of the small amount of RAM available. Note that the filesystem is decompressed out of Flash into RAM at boot time.

```
root@OpenWrt:~# df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/root                 3264      2408       856  74% /
none                     15276        16     15260   0% /tmp
```

Only a modest (850kB) amount of space remains available on the root (Flash) filesystem. In practice it was found that packages could be added to the root filesystem beyond the free space shown because the filesystem is compressed.

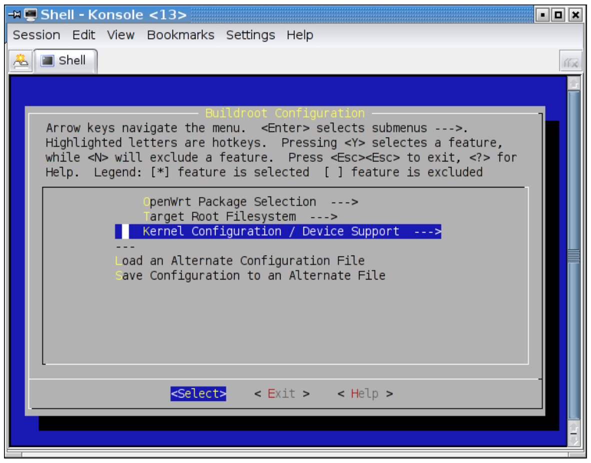
The /tmp filesystem is a 16MB RAM disk - as it is used MemFree is reduced by the corresponding amount.

5.4.2 Customisation

Having established the baseline of being able to build and flash OpenWRT onto the hardware the author began work on adding the pieces of software required to provide the additional functionality needed for our project.

As discussed in section 5.3 we will require both some kernel changes and additional user space packages to round out our software stack on the system.

OpenWRT uses a top level `.config` file and user interface tools taken from the "old" Linux kernel build system - for anyone that has built kernels in this era the interface will be familiar (Figure 5.1). This menu system allows you to select which of the standard packages will be built and installed in the flash image as well as make some general changes to the kernel.



```
Buildroot Configuration
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> selectes a feature,
while <N> will exclude a feature.  Press <Esc><Esc> to exit, <?> for
Help.  Legend: [*] feature is selected  [ ] feature is excluded

          OpenWrt Package Selection  --->
          Target Root Filesystem  --->
          Kernel Configuration / Device Support  --->
      ---
      Load an Alternate Configuration File
      Save Configuration to an Alternate File

        <Select>    < Exit >    < Help >
```

Figure 5.1: OpenWRT "buildroot" configuration screen

The standard package suite is quite comprehensive providing everything from samba to Asterisk. The AX25 tools are unfortunately not among them so it will be necessary to build these manually. The OpenWRT Wiki has a good level of information about this process.

Customising the kernel to include the require USB and sound functionality is straight forward - on the development host starting in the top level of the OpenWRT build tree;

```
cd ./build_mipsel/linux-2.4-brcm/linux-2.4.30/

make menuconfig
```

You are then presented with a standard kernel configuration menu.

If you select features to be built into the kernel (rather than as modules) they will automatically end up in the final binary firmware image. During development however the author found it easier to build the required support as modules and use `scp` to copy them onto the target system. The OpenWRT website also details how to add kernel modules into the "proper" packaging system but at the time of writing the author had not had an opportunity to explore this.

As a first step the appropriate sound modules were built and copied over the the WL-500G. In order for them to be loaded at boot a file was required in `/etc/modules`:

```
root@OpenWrt:/dev# less /etc/modules.d/72-snd-usb-audio
soundcore
snd-page-alloc
snd
snd-rawmidi
snd-hwdep
snd-timer
snd-pcm
snd-mixer-oss
snd-pcm-oss
snd-usb-lib
snd-usb-audio
```

5.4.3 Customisation Results

Rebooting with this file in place and logging in to the router again;

```
Root@OpenWrt:/dev# lsmod
Module                  Size  Used by    Tainted: P
snd-usb-audio          51944   0
snd-usb-lib            11380   0 [snd-usb-audio]
snd-pcm-oss            40372   0
snd-mixer-oss          14632   0 [snd-pcm-oss]
snd-pcm                67012   0 [snd-usb-audio snd-pcm-oss]
snd-timer              18632   0 [snd-pcm]
snd-hwdep               6008   0 [snd-usb-audio]
snd-rawmidi            17352   0 [snd-usb-lib]
snd                    42840   0 [snd-usb-audio snd-usb-lib snd-pcm-oss
                                  snd-mixer-oss snd-pcm snd-timer snd-hwdep
                                  snd-rawmidi]
snd-page-alloc          6688   0 [snd-usb-audio snd-mixer-oss snd-pcm
                                  snd-timer snd-hwdep snd-rawmidi snd]
soundcore               4048   3 [snd]
uhci                   30164   0 (unused)
usbcore                74792   1 [snd-usb-audio snd-usb-lib uhci]
wlcompat               14896   0 (unused)
wl                    423640   0 (unused)
switch-robo             4444   0 (unused)
switch-core             4896   0 [switch-robo]
diag                    3320   0 (unused)
```

Plugging in one of the USB Audio adaptors looked encouraging;

```
root@OpenWrt:/dev# lsusb
Bus 002 Device 001: ID 0000:0000
Bus 001 Device 001: ID 0000:0000
Bus 001 Device 002: ID 0d8c:000c C-Media Electronics, Inc.
```

Finally, the following yielded a suitable splat from the connected speakers;[5]

```
root@OpenWrt:/dev# cat /etc/passwd > /dev/sound/dsp
```

5.4.4 Building Hello World!

OpenWRT provides some documentation on how to compile your own applications in ./docs/buildroot-documentation.html so the next logical step seemed to be to try and build the usual hello world application.

On the host side building the app and copying it to the target was uneventful;

```
$ ../openwrt/staging_dir_mipsel/bin/mipsel-linux-uclibc-gcc -o hello hello.c
$ ls -l
total 12
-rwxr-xr-x  1 hugh hugh 7507 2006-06-01 22:10 hello
-rw-r--r--  1 hugh hugh  101 2006-06-01 22:09 hello.c
$ scp hello root@192.168.73.215:
root@192.168.73.215’s password:
hello                                 100% 7507     7.3KB/s   00:00
$
```

And running it on the target equally so;

```
root@OpenWrt:~# ./hello
Hello world!
```

5.4.5 Concluding Remarks

Unfortunately at this point other commitments intervened time-wise and we were unable to progress further before the due date of this report. Chapter 6 discusses planned next steps.

[5]What I had thought to be a permissions problem turned out to be nothing more than using the wrong device - `/dev/dsp` instead of `/dev/sound/dsp`.

Chapter 6

Conclusions

An overview of the general design considerations for a base band data radio has been presented. As this was not the focus of the authors work it is relatively brief.

A detailed look at re-purposing consumer hardware running Linux has been given along with coverage of the required software stack as both would apply to the project.

6.1 Next Steps

The next steps to be taken are largely software oriented;

- Investigate the next release of OpenWRT (kamikaze) - it is still a development only branch but may be stable enough for our needs and appears to be based on a 2.6.x kernel.
- Compile and install the ax25 kernel modules.
- Compile and install the ax25 utils package.
- Compile and install the soundmodem package.
- Do trial run of soundmodem using regular 1200 baud FSK modem and confirm correct off-air reception of local amateur packet radio signals. Make note of CPU usage. Use handheld radio to avoid issues with earth referencing on USB sound interface.
- Confirm correct transmission of sample packets at 1200 baud, again check CPU usage.
- Repeat above process using soundmodem's 9600 baud FSK implementation and look at CPU usage.
- Assuming on air tests are successful and the CPU utilisation looks ok, create proper package files for all additional software and note availability on OpenWRT wiki

With the above steps completed successfully we can now be confident that the software side of things is as it should be, and that the router possesses sufficient CPU power to handle the DSP functionality.

We can then turn to hardware modifications to integrate the router into the overall radio system enclosure.

6.2 Future work

While his formal involvement with the project comes to a close with the end of the ENGN3214 course, the author hopes to remain involved with ongoing work in this area as a matter of personal interest.

Integrating the re-purposed hardware into the final radio system is application specific and the required modifications have already been called out for the most part.

The author intends modifying the router to act as an integrated AX25 packet radio gateway that can be directly connected to a standard 2m band amateur transceiver. This will necessitate building the USB audio dongle into the unit, providing isolation on the audio outputs (transformers are simple but effective here) and providing a PTT output - an opto isolator will be used for this.

Appendix A

Appendix

A.1 Other Sources

The following books, papers and articles were read during the course of my work on this project. While they are not directly referenced, I would none the less like to acknowledge their influence.

- The ARRL Handbook for Radio Amateurs [Danzer 1998]
- Practical Costas loop design [Feigin 2002]
- Modern Digital and Analog Communication Systems [Lathi 1998]
- Writing for Computer Science [Zobel 2004]

Bibliography

BEECH, W. A., NIELSEN, D. E., AND TAYLOR, J. Version 2.2 Revision 1998. *AX.25 Link Access Protocol for Amateur packet radio*. Tuscon Amateur Packet Radio Corporation. (p. 4)

BORG, G. 2006. Telecommunications Systems ENGN3214 course materials. (pp. 1, 5)

C-Media Electronics Inc. 2004. *CM108 Highly Integratied USB Audio I/O Controller Datasheet V1.5*. C-Media Electronics Inc. (pp. 13, 15)

DANZER, P. Ed. 1998. *The ARRL Handbook for Radio Amateurs*. The American Radio Relay League. (p. 27)

Fairchild Semiconductor. 1999. *MM74HC4046 CMOS Phase Locked Loop Datasheet*. Fairchild Semiconductor. (p. 6)

FEIGIN, J. 2002. Practical Costas loop design. *RF Signal Processing*. (p. 27)

LATHI, B. P. 1998. *Modern Digital and Analog Communication Systems*. Oxford University Press. (p. 27)

Phillips Semiconductors. 1997. *74HC/HCT4046A Phase-locked-loop with VCO Datasheet*. Phillips Semiconductors. (p. 6)

Rabbit Semiconductor. 2000. *RabbitCore RCM2100 C-Programmable Module with Ethernet User's Manual*. Rabbit Semiconductor. (p. 10)

VARIOUS. 2006. Wikipedia, the free encyclopaedia.

ZOBEL, J. 2004. *Writing for Computer Science*. (p. 27)