Packetsniffer (802.11 b) for evaluation of receive signal strength

03.08.2001

Victorin Kagoué
Alexander Kordecki

Betreuer: Martin Kubisch

TU Berlin, TKN (Telekommunikations Network Group)

Inhalt

Einleitung

Zu Meßzwecken haben wir eine Software entworfen, die uns die Möglichkeit bietet, Daten zwischen dem in der TU entworfenen RaDevil-System (Radio Modem MAC Developing Platform) und einem Linux-Rechner zu übertragen.

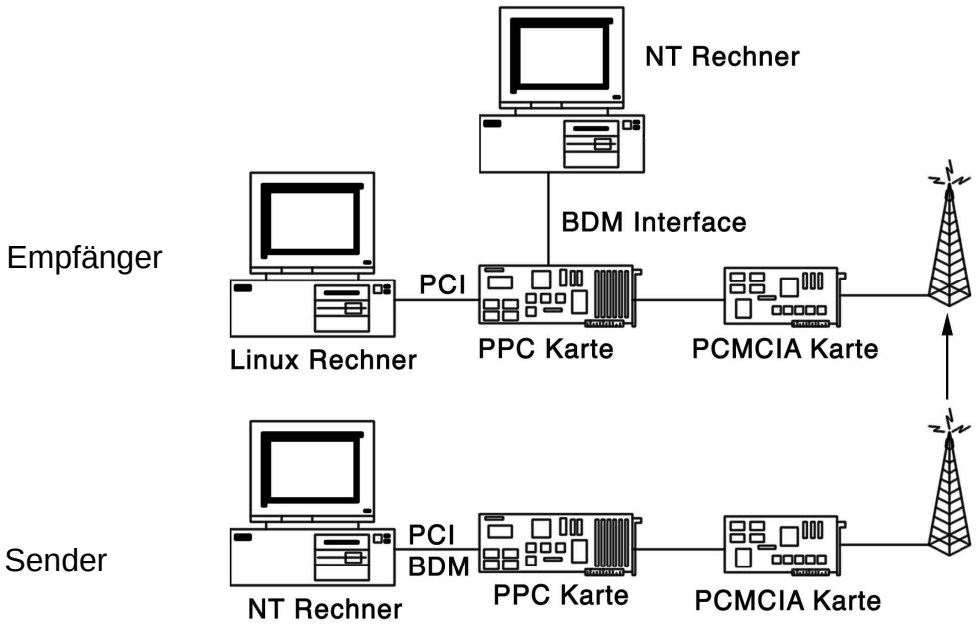
Die Software besteht, im Grunde genommen, aus drei Teilen, einem Kernel-Modul für Linux, einem Interface in der alten RaDevil Software und einigen Beispielsapplikationen um auf die Daten zugreifen zu können.

RaDevil

Das RaDevil System besteht aus einer Hardware- und einer Softwarekomponente. Als Hardware wird eine PCI-Karte mit einem Motorola PPC 860 Embedded System eingesetzt, auf der dann die dazugehörige Software läuft. Zusätzlich enthält das System noch eine Radio Modem PCMCIA Karte, auf die der PPC zugreifen kann.
Die Besonderheit der Karte ist, dass sie keinen MAC-Prozessor besitzt und somit die Möglichkeit bietet, alle beliebigen Signale direkt zu erhalten und auszuwerten.

Das System ermöglicht die experimentelle Auswertung von neuen MAC-Protokollen und die Messung protokollspezifischer Parameter.

Aufbau der Messumgebung



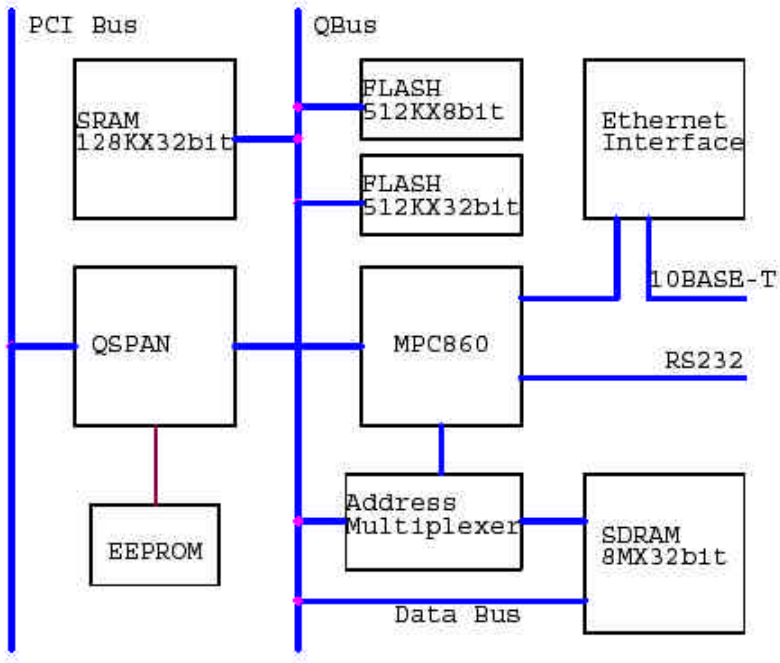
Beschreibung und Arbeitsweise

Die untere Hälfte des Bildes stellt den Sender dar. Hier benutzen wir das original RaDevil-System. Der Sender besteht aus einem NT-Rechner mit einer PPC-Karte auf dem PCI-Bus, darauf einer PCMCIA-Karte mit Antenne.

Die obere Hälfte zeigt den Empfänger bestehend aus einem Linux-Rechner mit der von uns entwickelten Software. Das System ist dem originalen RaDevil sehr ähnlich. Der Unterschied besteht darin, daß die PPC Karte jetzt auf dem PCI-Bus des Linux-Rechners sitzt, weiterhin aber ein NT-Rechner über das BDM-Interface mit der PPC Karte verbunden ist.
Das BDM-Interface dient der Programmierung und des Debuggings der PPC Karte. Hierfür nutzen wir die Software "Code Warrior for Embedded Systems", die nur unter NT läuft.

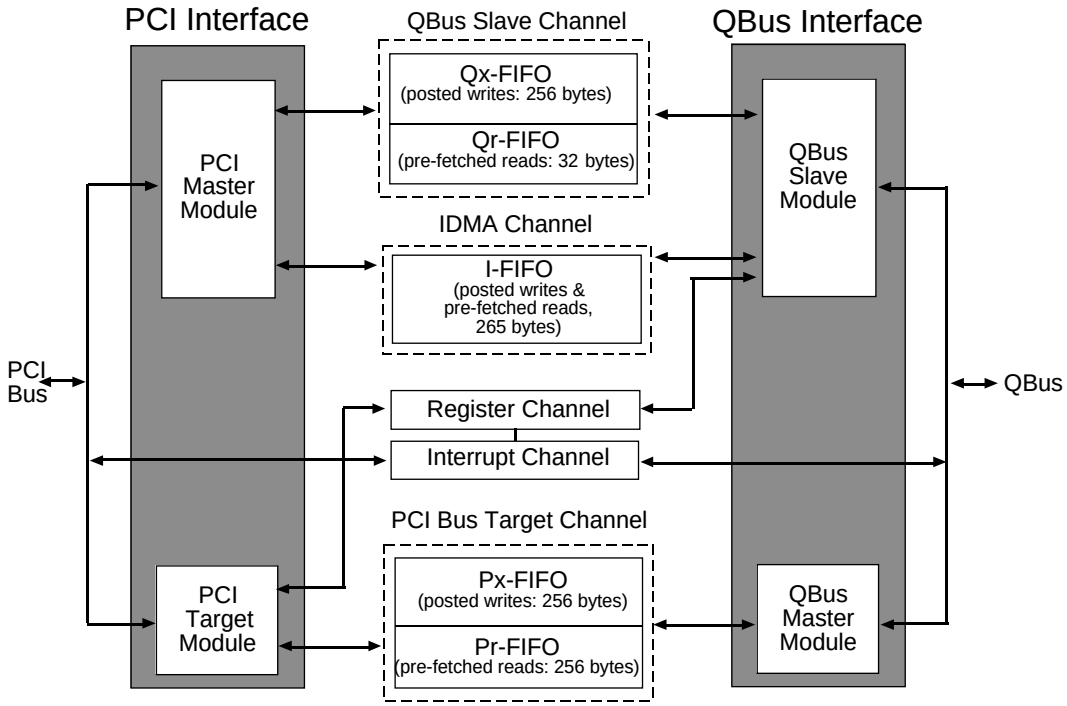
Die PPC-Karte

Die für uns interessanten Teile der Karte sind die beiden Busse, der PCI Bus und der QBus, sowie der Motorola PPC 860 Prozessor und der QSPAN Chip von Tundra, der eine Brücke zwischen dem PCI Bus und dem QBus bildet.
Zusätzlich sitzt auf dem QBus noch die PCMCIA Karte mit deren Hilfe wir Senden und Empfangen können.



Der QSPAN

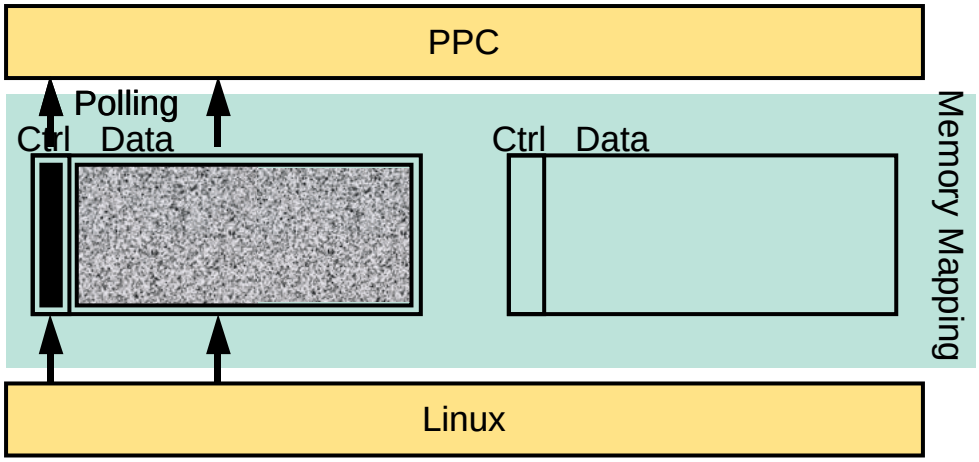
Der QSPAN hat zwei Interfaces, eines zum PCI Bus und eines zum QBus, zwischen diesen Interfaces kann er mit Hilfe von FIFOs Daten übertragen. Er übernimmt das Übersetzen von Interrupts und Speicherbereichen. Er kann von beiden Bussen aus programmiert werden.



Die Datenübertragung zwischen Linux & PPC

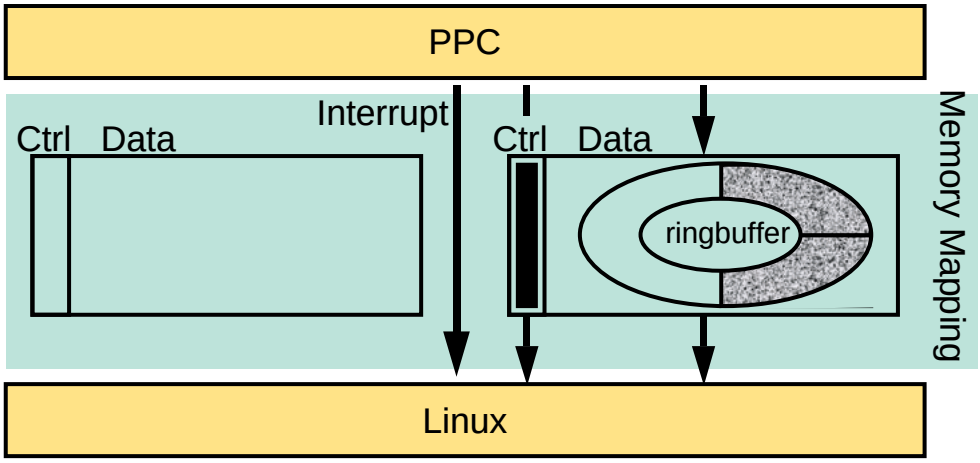
Zur Übermittlung der Daten zwischen dem Linux-Rechner und dem PPC haben wir uns für unterschiedliche Übertragungsmechanismen für die beiden Kommunikationsrichtungen entschieden, da wir mit dem Rechner eigentlich nur Daten empfangen möchten und nicht senden müssen.
Wir haben mit Hilfe des QSpan zwei Speicherbereiche des Linux-Rechners in den Speicher der PPC-Karte gemapped. Sie stellen unseren bidirektionalen Kommunikationskanal dar, für jede Richtung ein Speicherbereich. Jeder Speicherbereich besteht aus einer Kontrollsemaphore und einem Datenspeicher.

Linux -> PPC:
Die Kommunikation läuft hier folgendermaßen ab: Wenn der Linux-Rechner etwas zu senden hat, prüft er als erstes, ob die Semaphore gesetzt ist. Falls sie gesetzt ist, wartet er bis sie wieder frei ist, ansonsten schreibt er seine Daten in den dafür vorgesehenen Bereich und setzt die Semaphore danach.



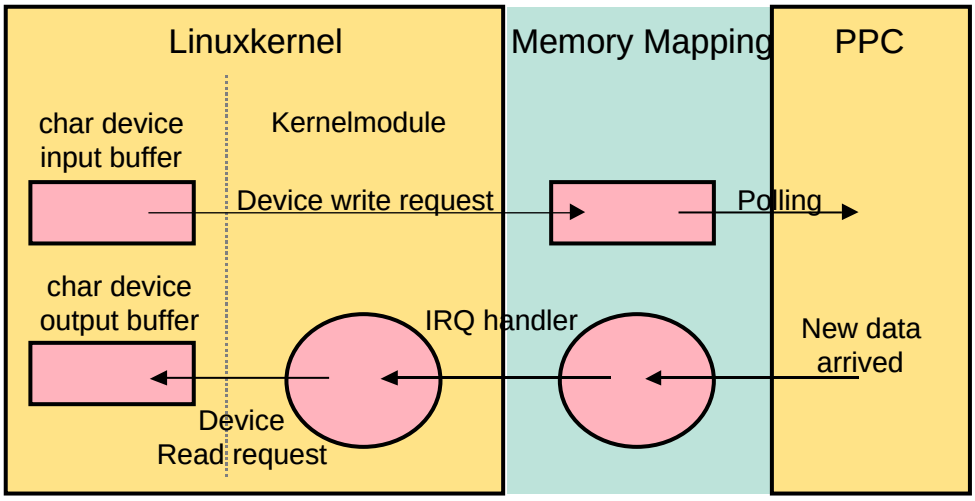
Der PPC pollt in regelmäßigen Abständen die Semaphore, um zu prüfen, ob neue Daten im Speicher stehen. Wenn sie gesetzt ist, liest er die Daten aus und löscht sie danach wieder.
Die Semaphore sorgt dafür, daß einerseits erkannt wird, wenn neue Daten vorliegen und andererseits noch nicht ausgelesene Daten nicht überschrieben werden.

PPC -> Linux:
In der anderen Richtung gestaltet sich der Aufbau etwas komplizierter, hier liegt ein Ringpuffer im gemappten Speicherbereich. Zusätzlich werden auch der Schreib- und Lesepointer geshared. Wenn Daten vom PPC an Linux gesendet werden sollen, wird erst einmal nachgeprüft, ob noch genügend Speicher im Ringpuffer frei ist. Falls ja wird der Datensatz in den Puffer geschrieben, falls nicht, wird er verworfen. Nach dem schreiben des Datensatzes wird geprüft, ob die Semaphore schon gesetzt ist, wenn ja, passiert nichts weiter, ansonsten wird mit Hilfe des QSpan ein Interrupt auf dem PCI-Bus ausgelöst.



Wenn der Linux-Rechner einen Interrupt von der Karte bekommt, werden alle Daten aus dem Ringpuffer ausgelesen und in einen eigenen Puffer geschrieben, danach wird die Semaphore gelöscht.

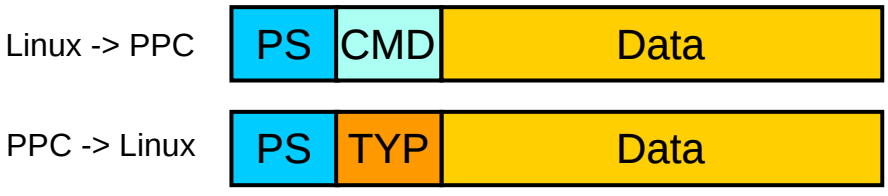
Puffermanagement



In diesem Schaubild ist noch einmal ein Überblick über die von uns verwendeten Puffer.
Das Kernelmodul besitzt noch einen eigenen Ringpuffer, in dem die vom PPC empfangenen Daten abgelegt werden. User-Space Applikationen können jetzt über ein char-Device mit dem Kernelmodul kommunizieren und hierüber Kommandos an den PPC absetzen, bzw. empfangene Daten aus dem Ringpuffer des Kernelmoduls auslesen.

Paketdatenformat

Das Paketdatenformat ist folgendermaßen aufgebaut:
Als erstes besitzt jedes Paket ein Längenfeld, daß die Länge des Paketes einschließlich des Header beinhaltet.
Als nächstes kommt ein Feld, daß je nach Übertragungsrichtung unterschiedlich genutzt wird, von Linux zum PPC steht hier der Befehl, der auf dem PPC aufgerufen werden soll, vom PPC zu Linux steht hier ein Pakettyp, um anzuzeigen, um was für ein Paket es sich handelt. Danach kommen in beiden Fällen zusätzliche Daten, entweder zum Befehl, oder bei einem empfangenen Paket der RSSI (Receive Signal Strength Indicator) und die Daten.



PS	Packet Size	PS	Packet Size
CMD	Command	TYP	Packettype
Data	Command Data	Data	Data

Befehlssatz / Pakettypen

Es existieren im Moment folgende Kommandos, die an die PPC-Karte übertragen werden können, bzw. Pakettypen, die gesendet werden:

Befehl	Code	Daten
initTransBuf	0x01	4 Byte Transferpuffer Adresse
echoCommand	0x02	Beliebige Daten, die zurückgeschickt werden
radevilCommand	0x03	Parameter für ein Radevil Kommando
restartCommand	0x04	keine Parameter

Pakettyp	Code	Daten
unknownCommand	0x00	Der übergebene Befehl
echoReply	0x02	Die Daten, die übergeben wurden
radevilData	0x03	1 Byte RSSI, danach die empfangenen Daten

Radevil Kommandos

Die Radevil Software unterstützt zur Zeit folgende Befehle:

Befehl	Code	Daten
DevOpen	0x00	Öffnet das Device für den Versandt/Empfang
DevSend	0x01	Sendet Datenpakete
DevStop	0x02	Hält den Datenversandt an
DevClose	0x03	Schliesst das Device

Module für den Linux Rechner

1. Das Kernelmodul "tundra.o":
   Hier ist der eigentliche Treiber und die Kommunikation mit der PPC Karte untergebracht
2. Das Perlscript "sendcommand"
   Ein Programm um beliebige Kommandos an die PPC Karte zu senden
   z.B. sendcommand 2 "hier sind die daten"
3. Das Perlscript "radevilcommand"
   Das Programm sendet ein Typ 3 Kommando an die PPC Karte
   z.B. "radevilcommand 0"
4. Das Perlscript "receive"
   Hiermit werden empfangene Pakete aus dem Device ausgelesen und ausgegeben.

Beispiel

Zur Verdeutlichung, welche Prozesse auf welchem Rechner gestartet werden müssen, vergeben wir den einzelnen Rechnern Namen.
LX: Linux-Rechner mit dem Kernelmodul
NT: NT-Rechner mit der Standard Radevil Umgebung
NL: NT-Rechner mit der Programmierumgebung der Karte im Linux-Rechner und dem Debuggingterminal für diese Karte

1. NL Debugging Terminal starten
2. NL CodeWarrior starten und Radevil-Linux in die Karte laden und starten
-> NL Auf dem Terminal sollte jetzt folgendes erscheinen:

```
Waiting for kernelmodule to connect ...
```

4. LX als root "insmod tundra.o" eingeben
-> NL Auf dem Terminal sollte erscheinen:

```
Kommando angekommen (typ: 1)
got transferbuffer-address (xxxxxxxx)
  ... Prism radio device driver ...
```

5. NT Debugging Terminal starten
6. NT CodeWarrior starten und standard Radevil-Software in die Karte laden und starten
-> NT Auf dem Terminal sollte erscheinen:

```
.
```

7. NT In der bash oder im DOS-Fenster "txctrl -dmatest" starten
-> NT Auf dem Terminal sollte erscheinen:

```
"Piper I" (Interrupt mode)
  ... Prism radio device driver ...
```

8. LX "radevilcommand 0" aufrufen
-> NL Auf dem Terminal sollte erscheinen:

```
Kommando angekommen (typ: 3)
 ... DevOpen ...
```

9. NT "txctrl -starttx" starten
-> NT Auf dem Terminal sollte erscheinen:

```
 ... DevOpen ...
```

10. NT noch mal "txctrl -starttx" starten
-> NT Auf dem Terminal sollte erscheinen:

```
 ... status | len last xmited ...
```

-> NL Auf dem Terminal sollte erscheinen:

```
schreibe daten laenge xxx in puffer groesse xxx ...
```

11. LX wenn es Probleme mit dem Empfang gibt (es passiert nichts mehr im NL Terminal), zusätzlich noch den Sender auf dem Linux-Rechner starten "radevilcommand 1"
-> NL es sollten Informationen über empfangene Pakete durchlaufen

12. LX "receive" starten

-> LX es wird ungefähr folgendes für jedes Paket auf dem Terminal ausgegeben:
```
New Packet - Length: xxx Type: 3 Data xx xx xx xx xx xx xx xx ...
```

Zum Beenden müssen folgende Befehle ausgeführt werden:

13: LX "sendcommand 4" starten, hiermit wird der PPC zurückgesetzt und das Kernelmodul wieder freigegeben
-> NL Auf dem Terminal sollte erscheinen:
```
Kommando angekommen (typ: 4)
...
Waiting for kernelmodule to connect ...
```

14: LX "rmmod tundra" starten, hiermit wird das Kernelmodul wieder aus dem Kern geladen.

Glossar

BDM	Background Debug Mode
FIFO	First In First Out - Stack
MAC	Medium Access Control
NT	New Technology (Windows NT)
PCI-Bus	Peripheral Component Interconnect Bus
PCMCIA	Personal Computer Menory Card International Association
PPC	Power PC
RaDevil	Radio Modem MAC Developing Platform
RSSI	Receive Signal Strength Indicator