GNU Radio
Wireless protocol analyses approach

Alex Verduin

July 2, 2008

Universiteit van Amsterdam

System and Network Engineering
Index

- Research question
- What is Software Defined Radio
  - Hardware
  - Software
- What is Radio Data System
- Approach
- Flowgraph of the RDS protocol
- Demo
- Conclusion and future work
- Questions
Research question

How can a system and network engineer use the USPR and GNU Radio to fulfill a wireless protocol analyses?
### SDR explained

**Figure:** Design principles of SDR

<table>
<thead>
<tr>
<th>SDR Target</th>
<th>Where</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimize</td>
<td>Antenna</td>
<td>USRP</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
<td></td>
</tr>
<tr>
<td>Maximize</td>
<td>Software</td>
<td>GNU Radio</td>
</tr>
<tr>
<td></td>
<td>Bits</td>
<td></td>
</tr>
</tbody>
</table>
My setup

Figure: My setup
Universal Software Radio Peripheral (USPR)

Figure: USRP
GNU Radio Software architecture

- GNU Radio Companion (GRC)
- Python
- SWIG
- C++
GNU Radio Software Architecture

GNU Radio Companion (GRC)

Python

SWIG

C++
Figure: GNU Radio Comagnion

>>> Warning: A connection can only be created between a source and an unconnected sink.
>>> Warning: A connection can only be created between a source and an unconnected sink.

Generating: "$/home/alex/helloworld.py$"
GNU Radio Software architecture

- GNU Radio Companion (GRC)
- Python
- SWIG
- C++
#!/usr/bin/env python

from gnuradio import gr
from gnuradio import audio

sampling_freq = 48000
ampl = 0.1

fg = gr.top_block()
src0 = gr.sig_source_f(sampling_freq, gr.GR_SIN_WAVE, 350, ampl)
src1 = gr.sig_source_f(sampling_freq, gr.GR_SIN_WAVE, 440, ampl)
dst = audio.sink(sampling_freq)
fg.connect(src0, (dst, 0))
fg.connect(src1, (dst, 1))
fg.start()
GNU Radio Software architecture

- GNU Radio Companion (GRC)
- Python
- SWIG
- C++
GNU Radio Software architecture

```c++
#include "gr_rds_freq_divider.h"

namespace gr {

// The private constructor.

gr_rds_freq Divider::gr_rds_freq Divider (int divider)
  : gr_sync_block ("gr_rds_freq Divider",
                   gr_make_io_signature (MIN_IN, MAX_IN, sizeof (float)),
                   gr_make_io_signature (MIN_OUT, MAX_OUT, sizeof (float)))
{
  d Divider = 0;
  DIVIDER = divider;
  d_sign Last = d_sign Current = false;
  d_out = 1;
}

/* The virtual destructor.
 */

gr_rds_freq Divider::~gr_rds_freq Divider ()
{
  // |
}

int
gr_rds_freq Divider::work (int noutput_items,
                           gr_vector const void * &input_items,
                           gr_vector void * &output_items)
{
  const float *in = (const float *) input_items[0];
  float *out = (float *) output_items[0];
  for (int i = 0; i < noutput_items; i++){
    d_sign Current = (in[i] > 0 ? true : false);

    if(d_sign Current != d_sign Last) {
      // A zero cross
      if(++d Divider == DIVIDER) {
        d_out * = -1;
        d Divider = 0;
      }
    }
  }
```

Radio Data System (RDS)

Figure: RDS logo

Figure: RDS example
Approach

- Understand the design of the USPR and GNU Radio
- Install and try code examples
- Study protocol specifications and search for existing GNU Radio code
- Create flow graph
- Create testbed
- Capture raw samples
- Analyse the protocol
RDS Flow graph 2/3

Source: File or USRP (1)

Channel Filter (2)

Guts (3)

FM Filter (5)

Audio Sink (4)
Lets do some demoing!!
Conclusions:

- The defined approach works.
- Writing code, easy with Python, even more with GRC, difficult in C++
- The SNE’er needs some radio and SDR knowledge.
- Not all protocols can be fully analysed.

Future work:

- Analyse more protocols
- Extend the research with transmitting
Questions:

- .......?