SOFTWARE DEFINED RADIO ARCHITECTURES EVALUATION

Alvaro Palomo Navarro
Rudi Villing
Ronan Farrell

Institute of Microelectronics and Wireless Communications (IMWS)
National University of Ireland, Maynooth (NUIM)

Outline

- SDR Architectures Overview
- Evaluation Methodology
  - Objectives
  - Software Loopback Structure
- Tests Results
  - Data Throughput
  - Computation profile
  - CPU Load
  - Memory Load
- Conclusions
Software Defined Radio Architectures Overview

- OSSIE and GNU Radio represent the two main open-source SDR architectures nowadays.

- Both of them are based on the same principle:
  - Waveform re-configurability.
  - Multiplatform implementation.
  - Standard component libraries.
Software Defined Radio
Architectures Overview

- GNU Radio:
  - Components implemented in C++.
  - Python is used to build up the flow graph.
  - One process with possibility of multiple threads. Direct function calls.

- OSSIE:
  - Components implemented in C++.
  - Assembling module using XML.
  - CORBA is used for multi-process communications.
Evaluation Methodology:

Objectives

- OSSIE and GNU Radio performance.
- Common hardware platform.
- Equivalent waveform.
- Full-duplex upper limit on throughput.
- No physical interfaces constrains.
- Analysis of the performance’s causes.
Evaluation Methodology:
Software Loopback Test Structure

- **Test-bed:** 3GHz Pentium 4 CPU and 1024 MB of RAM.
- **Operating system:** Ubuntu 7.1
- **Framework versions:**
  - OSSIE 0.6.2
  - GNU Radio 3.1.2
- **Main factors:**
  - Low-complexity waveform.
  - Components availability.
- **Information generation:**
  - **OSSIE:**
    - Pre-allocated data block.
    - Fixed packet size of 64 MB.
  - **GNU Radio:**
    - Dynamic generation of the packets using the packet number.
    - The size can be selected.
Test Results:

CPU Load

- CPU monitoring using the Linux `top` command for data transmission.

- Fixed packet size: 64 MB.

- Both frameworks use roughly the **100%** of the CPU capacity.
  - GNU Radio with one single process.
  - OSSIE individual components: 10 to 20% of CPU load each.
  - No source of non-computation delays (i.e. interfaces with RF front-ends).

- Reasonable to assume linearity between the performance and the processor's speed.
  - GPP vs. Embedded
Test Results:

Memory Load

- **exmap** tool for memory monitor: suitable for virtual and share memory analysis.

- Virtual memory: total amount requested by a process but not fully used.

- Shared memory (i.e. common libraries): Each N-process uses 1/N.

- Both applications could run within a 32 to 64 MB memory.
Test Results:

Data Throughput

- Estimation of the maximum full-duplex throughput achievable for each framework.
- Fixed amount of information (10 MB) vs Packet size.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Packet Size (bytes)</th>
<th>Throughput (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSSIE</td>
<td>64</td>
<td>0.72</td>
</tr>
<tr>
<td>GNU Radio</td>
<td>64</td>
<td>0.59</td>
</tr>
<tr>
<td>GNU Radio</td>
<td>256</td>
<td>0.68</td>
</tr>
<tr>
<td>GNU Radio</td>
<td>1024</td>
<td>0.71</td>
</tr>
</tbody>
</table>

Initial conclusions
- Performance equal (and not very good) in both
- Reasonable to assume double performance for half-duplex radio.
- More realistic waveform would lead to lower performance.
- In GNU Radio part of the code is written in Python -> Maybe explanation for the performance.
- CORBA used in OSSIE -> Worse performance than GNU Radio could be expected.
Test Results:

**Computation Profiling**

- Why is maximum bit rate rather slow?
  - Profile to look for computation hotspots
  - Easy Python profiling using `cProfile`.

- Small difference for fixed amount of data sent.

- Dramatic increase for a fixed amount of packets and size variations.

- Results approx. linear with amount of data sent
  - data movement/copying takes most time

![Processing Time for GNU Radio](image-url)
Test Results:

Computation Profile

- Data moving management made by memory copying:
  - “work()” method.
  - Modules with different I/O types: array copying element by element.
  - Modules with equal I/O types: “memcpy” function.

- Possible improvement: Pointers manipulation instead of memcpy when input and output data are identical.
Conclusions

- Deeper profiling and further optimization still possible
- Loopback test is a reasonable estimator of upper bound throughput performance.
- Can expect real SDR applications to achieve lower performance.
- Maximum throughput achieved was around 700 kbps for both frameworks:
  - OSSIE slightly faster.
  - Surprisingly low.
  - Reasonable to assume linear improvement with a processor speed increment.
- Smaller granularity vs. Software Radio
Thank you
Any questions?
apalomo@eeng.nuim.ie