Embedded SDR for Small Form Factor Systems

Philip Balister, Tom Tsou, and Jeff Reed
MPRG
Wireless @ Virginia Tech
Blacksburg, VA 24060
balister@vt.edu
Outline

- Embedded Software Defined Radio
- SDR Frameworks
  - Software Communication Architecture (SCA)
  - OSSIE
- Packet data transmitter
- Performance measurements (profiling)
- Memory Usage
- Message transfer times
- Conclusions and Questions
Embedded SDR

- Real Radios are “small”
  - Have you ever seen a radio the size of a PC?
- SWAPO
  - Size, weight, and power
- Memory usage increases power and cost
- Power consumption increases with clock speed
- Framework overhead is an issue
Why Frameworks for SDR?

- Frameworks provide structure for a SDR
  - Improve component re-use
  - Speed component development
  - Reduce developer training requirements

- Framework costs
  - More memory
  - More clock cycles
  - More power
OSSIE

- Open Source SCA Implementation::Embedded
- Written in C++
  - Current development focused on Linux
  - Should work on other UNIX based OS'
- Uses OmniORB and XERCES
- First released in July 2004
- Development team composed of VT students
  - Graduate and under graduate students
- Project web page http://ossie.mprg.org
OMAP Starter Kit (OSK) and Universal Software Radio Peripheral (USRP)
USRP Capabilities

- Developed for the GNU Radio project
- A/D's operate at 64 MSPS @ 12 bits
- D/A's operate at 128 MSPS @ 14 bits
- FPGA provides digital down conversion
  - Sample rate reduction
  - Receiver tuning
- TX tuning and sample rate conversion done in AD9862
What to measure

- Framework overhead
- Processor Usage
  - Power
- Memory usage
  - Determines amount of memory needed
- Inter-component message time
  - Sets performance limits (latency)
  - Guides design decisions
Test waveform
Processor Usage

- Measure time spent in components
  - Record time per function
- Compare “real work” versus CF
- Data collected on Pentium
  - Better support for profiling
OProfile

- OProfile is [http://oprofile.sf.net](http://oprofile.sf.net)
- Based on timers or HW counters
- Profiles the entire system, including kernel
- Provides detailed source annotation
  - May go down to assembly code level
- Call Graphs show time allocated to calling routines
Test Waveform Processor Usage
RandomBits Component

- Libc
- LibomniORB4
- Libpthread
- LibstandardInterfaces
- RandomBits
- Other
Modulator Component
Memory Usage

- Operating system determines memory usage
  - Memory management units (MMU)
- DSP's and Real Time OS's may not provide memory management
- MMU enables shared code between processes
- This subject is very complex
- Data collected on the OSK
Memory usage perspective

- JTRS waveform memory usage
  - 80 – 100 MB RAM
  - 245 - 275 MB Flash
  - Milcom 2006, Hasan, Jensen, etal. Designing Software Defined Small Form Fit Radios For JTRS Networking
Some terms

- Pages – Units of memory (4K)
  - Read only, read/write
- Shared Libraries
- Swap
- Dirty pages
- Demand loading
Memory Manager

Process A memory
- Code A
- Library 1
- Library 1
- Stack A
- Data A
- Data A
- 6 Pages

Physical Memory
- Code A
- Code B
- Library 1
- Library 1
- Stack A
- Stack A
- Data A
- Data A
- Stack B
- Data B
- Library 2
- Library 2
- 11 Pages

Process B memory
- Code B
- Library 1
- Library 1
- Stack B
- Data B
- Library 2
- Library 2
- 7 Pages
## Shared Library memory usage

<table>
<thead>
<tr>
<th>Library</th>
<th>Code(K)</th>
<th>Variables(K)</th>
<th>Constants(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xerces-c</td>
<td>3436</td>
<td>200</td>
<td>8</td>
</tr>
<tr>
<td>OmniDynamic4</td>
<td>1768</td>
<td>204</td>
<td>0</td>
</tr>
<tr>
<td>OmniORB4</td>
<td>1340</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>Libc</td>
<td>1056</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Ossieidl</td>
<td>952</td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td>Stdcc++</td>
<td>684</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>Libm</td>
<td>640</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>StandardInterface</td>
<td>460</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>Ossiecf</td>
<td>396</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>Ossieparser</td>
<td>228</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>ld-2.5</td>
<td>108</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Pthrend</td>
<td>72</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Libgcc</td>
<td>40</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Nssfiles</td>
<td>36</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Omnithread</td>
<td>16</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
## Component memory usage

<table>
<thead>
<tr>
<th>Component</th>
<th>Code(K)</th>
<th>Heap(K)</th>
<th>Stack(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>USRP</td>
<td>100</td>
<td>2472</td>
<td>49212</td>
</tr>
<tr>
<td>Modulator</td>
<td>48</td>
<td>348</td>
<td>41024</td>
</tr>
<tr>
<td>Interpolator</td>
<td>48</td>
<td>348</td>
<td>41024</td>
</tr>
<tr>
<td>Random Bits</td>
<td>44</td>
<td>348</td>
<td>41024</td>
</tr>
<tr>
<td>USRP TX Cont</td>
<td>44</td>
<td>348</td>
<td>24648</td>
</tr>
</tbody>
</table>

- OmniORB creates 10MB thread stacks
- One thread per CORBA servant
## Overall Memory Usage

<table>
<thead>
<tr>
<th>Process</th>
<th>Virtual</th>
<th>Resource</th>
<th>Shared</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeBooter</td>
<td>45564</td>
<td>7736</td>
<td>6448</td>
<td>0</td>
</tr>
<tr>
<td>USRP</td>
<td>64352</td>
<td>7288</td>
<td>5820</td>
<td>1</td>
</tr>
<tr>
<td>Interpolator</td>
<td>53840</td>
<td>6584</td>
<td>5720</td>
<td>30.2</td>
</tr>
<tr>
<td>Modulator</td>
<td>61984</td>
<td>6532</td>
<td>5680</td>
<td>1.6</td>
</tr>
<tr>
<td>RandomBits</td>
<td>61980</td>
<td>6520</td>
<td>5668</td>
<td>0.3</td>
</tr>
<tr>
<td>USRP_TX_Control</td>
<td>37404</td>
<td>6472</td>
<td>5652</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Motivation for Latency Profiling

• Determine runtime end-to-end waveform latencies
• Analysis under different environments and configurations
• Detect unanticipated system behavior
• Improve performance and reliability
• Optimal buffer sizes
Latency Monitoring Requirements

- Accurate and precise timing information
  - µs or better resolution
- Minimally intrusive
  - Efficient operation with low CPU and I/O overhead
- Portability and code reuse
  - Functionality on multiple operating systems and architectures
  - Architecture dependent operations should be avoided if possible
Target Systems

- Typical COTS PC
  - Uniprocessor 3.2 GHz Pentium IV
  - 2GB RAM
- OMAP OSK
  - 192MHz ARM
- Operating System
  - Linux 2.6.20 with PREEMPT_RT patch by Ingo Molnar
High Resolution Clock Sources

- **PC**
  - Time Stamp Counter (TSC)
    - Free running 64-bit counter operating at CPU clockrate (3.2GHz)
    - Accessible through x86 assembly operation *RDTSC*

- **OMAP**
  - Microprocessor Unit (MPU) Timer
    - 32-bit configurable clock
    - Accessed through special timer register
gettimeofday()

- Linux system call
  - Lets kernel manage timekeeping clocksources and cycle conversions
  - Allows for architecture independent interface
- High resolution
  - μs accuracy (if hardware allows)
- Acceptable overhead
  - Low system call overhead compared to overall waveform latency
  - < 1μs mean offset from using RDTSC operand
Integrating with OSSIE SCA

- Simple timestamp scheme
- Record a fixed number of timestamps at selected endpoints
- Disk I/O and timing analysis performed offline after data collection (MATLAB)
- Reusable code
  - Timing implementation still dependent on underlying kernel and architecture
Measurement Objectives

- Effects of real-time scheduling
- FIFO and Round-Robin schemes
- Skeleton waveform and functional FM application
Generic Waveform

- Component threading and interface overhead
- CORBA latency between co-located components
- Buffering delay
- No signal processing
PC - Waveform Latency
OMAP – Waveform Latency
FM Waveform

- Operational narrowband transmitter
- Signal processing in modulation and filter operations

Diagram:
- Audio Source
  - FM Modulator
    - FIR Interpolator
    - USRP
      - FIR Interpolator
FM Waveform Latency

Graph showing measured latency (ms) over sample number with two lines representing Default Scheduler and RT-FIFO Scheduler.
Future Work

- Timing details for dual-core and SMP architectures
  - Active topic in Linux kernel development
- Handling of distributed waveforms
  - No readily available precise synchronized time source
- Behavior under specific system loads
- Tool chain integration
Conclusions

- CORBA not a problem for data transfer
  - At least for CPU cycles
- Framework overhead low for components with significant processing
- Need to evaluate waveform start up
  - Evaluate XML parsing
- Real time kernel improves consistency
Thanks!

- National Institute of Justice
- National Science Foundation
- Wireless@VT Affiliates
Questions?