Implementation of OSSIE on OMAP

M. Robert
P. Balister
J. Reed

June 23, 2006
Implementation of OSSIE on OMAP

Max Robert
Philip Balister
Jeffrey H. Reed

Wireless @ Virginia Tech
Mobile and Portable Radio Research Group (MPRG)
max.robert@gmail.com
philip@balister.com
Ossie on OMAP

- OSSIE provides SCA for NIJ Chamelonic radio
- OMAP OSK Description
  - USRP used for RF interface
- Issues building and running OSSIE on TI OMAP processor
  - Building OSSIE for the OMAP
  - Memory Management
  - Software Profiling
OMAP Starter Kit (OSK)

- OMAP 5912 Starter Kit (OSK)
  - Available from [http://www.ti.com](http://www.ti.com)
  - $295
- LCD Display available
- OSK board uses 5 volts @ 250 mA
OMAP Block Diagram
OMAP Capabilities

- ARM926EJ-S Core operating at 192 Mhz
- C55X DSP Core operating at 192 Mhz
- USB 1.1 controller
- Numerous interfaces
  - GPIO
  - I2C
  - LCD Controller
USRP

☐ 4 A/D
☐ 4 D/A
☐ FPGA
☐ USB
☐ Daughter Boards
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
  - RX tuning
- TX tuning and sample rate conversion done in AD9862
USRP Daughter Boards

- RFX series boards
  - Support transmit and receive
  - Several models supporting different bands
    - 400-500 Mhz, 900-1000 Mhz, 2.4-2.5 Ghz
  - DBSRX receive only, 800-2400 Mhz
  - Basic TX and RX boards, provide AC coupled inputs. Can bandpass sample to 200 Mhz
  - DC coupled versions available for LF operation
OSK and USRP
Building OSSIE for the OSK

- Development environment
  - Open Embedded versus Buildroot
- Supporting Software
  - omniORB
  - xerces-c
Development Environment

- OSSIE runs on any hardware using Linux
  - Not uCLinux
- Cross compiler required for ARM processor
- Need cross compiled version of libraries
- Produce images of file systems suitable for writing into flash memory
- Need a way to automate this process!
Some terms

- **Package**
  - Installable piece of software
    - rpm, ipk, deb etc

- **Image**
  - Something installed directly onto hardware
    - Flash file system images
    - kernel images
Where to start?

- **Crosstools** ([http://www.kegel.com/crosstool/](http://www.kegel.com/crosstool/))
  - Generates cross tool chains

- **Buildroot** ([http://buildroot.uclibc.org/](http://buildroot.uclibc.org/))
  - Generates tools chains and creates file systems
  - Easy to set up

- **OpenEmbedded** ([http://www.openembedded.org/](http://www.openembedded.org/))
  - Generates tool chains and creates file systems
  - Harder to set up, but more flexibility
Why Open Embedded

- Easy to support multiple machines
- Large package database
- Handles complex software dependencies
- Strong developer and user community
- Simple to use, if you are a genius
Open Embedded Basics

- Open Embedded meta data
  - Machine configuration
  - Distribution configuration
  - Package build data (bitbake files)

- Bitbake
  - Processes the meta data
  - Produces desired output, such as images and packages
Machine Configuration

- Kernel details
- Required kernel modules
- Flash sizes
- Hardware details (such as serial ports)
- Processor specific compiler options
- Machine specific software
  - such as specific display server packages
Distribution Configuration

- Define software versions
  - compilers
  - system libraries
Bitbake files

- Package information
  - description, license, maintainer, etc
- Source location
- Required bitbake modules
  - autotools, pkgconfig, etc
- Specialized build scripts
DESCRIPTION = "SWIG - Simplified Wrapper and Interface Generator"
HOMEPAGE = "http://swig.sourceforge.net/
LICENSE = "BSD"
SECTION = "devel"

SRC_URI = "${SOURCEFORGE_MIRROR}/swig/swig-${PV}.tar.gz"
S = "${WORKDIR}/SWIG-${PV}"

inherit autotools

EXTRA_OECONF = "--with-python=${STAGING_BINDIR} --with-
    swiglibdir=${STAGING_DIR}/${BUILD_SYS}/swig"

do_configure() {
    oe_runconf
}

Complex bb file

```bash
do_compile () {
    export XERCESCROOT=${S}
    cd src/xercesc
    # runConfigure is going to bust CC and CXX I bet
    CC_SAVE="${CC}"
    CXX_SAVE="${CXX}"
    ./runConfigure -plinux -c${CC} -x${CXX} -minmem -nsocket -tnative -rpthread
    CC="${CC_SAVE}"
    CXX="${CXX_SAVE}"
    oe_runmake
}

do_stage () {
    oe_libinstall -C lib libxerces-c ${STAGING_LIBDIR}
    oe_libinstall -C lib libxerces-depdom ${STAGING_LIBDIR}
    cp -pPR include/xercesc ${STAGING_INCDIR}
}

do_install () {
    oe_libinstall -C lib libxerces-c ${D}${libdir}
    oe_libinstall -C lib libxerces-depdom ${D}${libdir}
}
```
Open Embedded Summary

- Challenging to get started with OE
- Build on other peoples work
- Document a packages build process
- OE is capable of building complex packages
- Bitbake files available for OSSIE
SCA Overhead

- SCA requires some overhead
  - Memory overhead
  - Framework overhead
    - CORBA for inter-component communication
- How can we measure the impact of the SCA
  - Analysis of component memory usage
  - Code Profiling of running waveforms
Memory Usage

- Memory Management Unit (MMU) critical
- Most libraries built as shared libraries
  - This dramatically uses physical memory usage
- MMU leads to real time issues
  - Demand loading pages from disk takes time
OSSIE/OMAP Benchmarks

- **Memory ramifications**
  - Typical baseline memory usage
    - 500 kB for omniNames
    - 1.5 MB for nodeBooter
    - 1 MB for component
  - Significant realtime issues
    - Cache management

- **Non-volatile memory**
  - Total file system
    - 38 MB
  - SCA-related disk usage
    - 2.6 MB

---

**Maximum memory usage**

<table>
<thead>
<tr>
<th>Name</th>
<th>k bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>omniNames</td>
<td>9328</td>
</tr>
<tr>
<td>nodeBooter</td>
<td>17856</td>
</tr>
<tr>
<td>Component average</td>
<td>20204</td>
</tr>
</tbody>
</table>
Profiling

- Use oprofile (http://oprofile.sourceforge.net)
- Can profile entire system, including kernel
- Provides detailed source annotation
- Instruction level profiling available
- Call graphs available
FM Receiver Test Waveform

- USRP converts baseband RF to samples
- Two devices
  - USRP Interface
  - Sound Card Interface
- Two components running in PC
  - Decimator
  - FM Demodulator
Decimator Profile

<table>
<thead>
<tr>
<th>samples</th>
<th>%</th>
<th>image name</th>
<th>symbol name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123395</td>
<td>84.2471</td>
<td>Decimator</td>
<td>fir_filter::do_work(bool, short, short&amp;)</td>
</tr>
<tr>
<td>19979</td>
<td>13.6405</td>
<td>Decimator</td>
<td>run_decimation(void*)</td>
</tr>
<tr>
<td>1577</td>
<td>1.0767</td>
<td>Decimator</td>
<td>dataIn_i::pushPacket(PortTypes::ShortSequence const&amp;, PortTypes::ShortSequence const&amp;)</td>
</tr>
<tr>
<td>286</td>
<td>0.1953</td>
<td>libc-2.3.6.so</td>
<td>memcpy</td>
</tr>
<tr>
<td>42</td>
<td>0.0287</td>
<td>libomniORB4.so.0.6</td>
<td>omniCallHandle::upcall(omniServant*, omniCallDescriptor&amp;)</td>
</tr>
</tbody>
</table>

- fir_filter::do_work has no CORBA content
- run_decimation loops over input data calling fir_filter::do_work and sends data via CORBA when done
- dataIn_i::pushPacket receives data from CORBA and makes a local copy
FM Demodulator Profile

- **FM_in_i::pushPacket** does all processing
- Reimplementing NCO using tables should reduce CPU usage
USRP Device Profile

- Need to look at data copying
- Inspection of mixed source/assembly suggests looking at CORBA sequences
Profiling Conclusions

- SCA and CORBA do not dominate CPU usage
- Need to review results carefully due to inlined functions
  - CORBA sequences are inlined
- Profiling very useful for showing bottlenecks
- Develop on a desktop and profile there before building for embedded target
Conclusions

- OSK board is a useful target for SCA development
  - Forces engineer to pay attention to resources
- NIJ Project Specific recommendations
  - Investigate reducing sampling rate coming from the USRP further by changing FPGA image
  - Study use of MCBSP serial ports on the OMAP s data can bypass USB interface.
  - Attempt to route MCBSP port directly to DSP
Acknowledgements

The following entities provided significant levels of funding for building OSSIE on the OSK:

- IC Postdoctoral Research Fellowship
- National Institute of Justice
- National Science Foundation
- SAIC