ALOE Framework and Waveform Design Workshop

Vuk Marojevic
Ismael Gomez
Antoni Gelonch

SDR – WInnComm – Europe 2013
Outline

1. Context
2. ALOE Concepts and Framework
3. Computing Resource Management
4. Waveform Development and Deployment
5. Conclusions
The software radio or SDR provides a flexible radio architecture that allows changing the radio personality, possibly in real-time.

Software

• **Waveforms (SDR applications)**
  • DSP algorithms
  • Radio’s physical layer behavior

• **Middleware (SDR framework)**
  • Software layer between applications and hardware
  • Execution environment for waveforms
  • Individual hardware and software development
  • Waveform loading and unloading \(\rightarrow\) reconfiguration
  • Portability and reuse of components
Hardware

TODAY

multicores

small clusters (heterogeneous)

DSP  GPP  FPGA

TOMORROW

many-cores
ALOE Context

UMTS

LTE

SDR Application 2
(Waveform 2)

SDR Application N
(Waveform N)

ALOE
SDR Computing

1. Multiprocessing
2. Lightweight
3. Platform Independence

ALOE Concepts

2004…

ALOE Tools

Open source
Heterogeneous Multiprocessing

SDR Platform

PE (GPP)  ALOE
PE (DSP)  ALOE
PE (DRA)  ALOE
(FPGA)  ALOE
PE (DRA)  ALOE
PE (DRA)  ALOE
PE (DRA)  ALOE
PE (DRA)  ALOE
PE (ASIP)  ALOE
PE (ASIC)  DAC
      ADC

PE with OS
PE without OS
ALOE services
Platform services

PE: processing element
DRA: dynamically reconfigurable area
Abstract Application Layer

Real Application Layer

ALOE Layer

Hardware Layer

ALOE VIRTUAL PLATFORM

ALOE

ALOE

ALOE

PE: Processing Element
ALOE Architecture

- DSP Module
- OESR API
- OESR
- RTDAL API

**OESR**: Operating Environment for Software Radio

**RTDAL**: Real-Time Distributed Abstraction Layer

**RTDAL POSIX implementation**
- Scheduler, file I/O, shm, timers

**RTDAL** implementation with no Operating System
- Full implementation of the scheduler, file I/O, shm, timers, etc.

**Hardware + OS kernel**
Real-Time Distributed Abstraction Layer (RTDAL)

- Interprocessor communication
- Synchronization
- Scheduling
  - pipelined execution, partitioned scheduling
  - 1 thread per processing core
- RTDAL API
  - Task creation and management
  - Interfaces
  - ADC/DAC abstraction
  - Time functions
Operating Environment for Software Radio (OESR)

- Automatic mapping of waveforms
- Location-transparent inter-module communications
- Configuration and visualization of variables & parameters
- Logs, counters, ...
Computing Resource Management
Wireless Communications Characteristics

- Continuous data transmission and reception
- Real-time services → real-time processing
- RAT/mode/QoS target → processing demands
- Heterogeneous multiprocessor platforms
- Limited computing resources
- Dynamic reconfigurations

RAT: Radio Access Technology
“Provide sufficient computing resource to waveforms for real-time processing”

Real-time constraints:

- Minimum throughput
- Maximum latency
Scheduling

Scheduling is the method by which threads, processes or data flows are given access to system resources. [Wikipedia]
# Static vs. Dynamic Scheduling (I)

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline (compile time), before execution</td>
<td>Online (runtime), at execution</td>
</tr>
<tr>
<td>Deterministic performance</td>
<td>Nondeterministic performance</td>
</tr>
<tr>
<td>Avoid migrations → less overhead &amp; fewer cache misses</td>
<td>Migrations → overhead &amp; cache misses</td>
</tr>
<tr>
<td>Avoid task locks → less system calls → less overhead</td>
<td>Task locks → more system calls → more overhead</td>
</tr>
<tr>
<td>Regular, periodic tasks with a priori information</td>
<td>Irregular, aperiodic tasks with unknown characteristics a priori</td>
</tr>
<tr>
<td>Runtime rescheduling costly</td>
<td>Easy to add new task at runtime</td>
</tr>
</tbody>
</table>
Static vs. Dynamic Scheduling (II)

- Scheduling overhead increases...
  - ...with the waveform granularity
  - ...with the number of processing elements
  - ...inversely to task execution time

- *Global-dynamic-preemptive* scheduling
  + flexible
  - may incur significant resource overhead

- *Partitioned-static* scheduling scales better

ALOE Resource Management

- **Pipelining**
- **Partitioned scheduling**: static and cooperative
  - Low overhead
  - Easy to implement
  - Scalable
- **Heterogeneous platforms** (w/o shared memory)
- Requires task-to-processor **mapping**
Data packet: $x$ samples

Packet arrival rate: $1/T$ Hz

Throughput requirement: $x/T$ samples/s

Process 1 input data packet every $T$ seconds
$t = 0$
Pipelining

\[ t = T \]
Pipelining

\[ t = 2T \]
Pipelining

\[ t = 3T \]
Pipelining

$t = 4T$

Processing latency: 4 time slots ($4 \cdot T$)

Processing throughput: 1 packet every $T$ seconds ($x/T$ input samples/s)
**Scheduling Example w/o Pipeline**

\[ \tau_i : \text{execution time per sample} \]

\[ \tau_1 = 0.1 \, \mu s \]
\[ \tau_2 = 0.5 \, \mu s \]
\[ \tau_3 = 0.1 \, \mu s \]
\[ \tau_4 = 0.1 \, \mu s \]
\[ \tau_5 = 0.1 \, \mu s \]

100 · \( \tau_2 \) = 50

**Processing latency:** 90 µs

**Latency:** \( f(\text{platform, mapping}) \)

**Processing latency:** 80 µs + tx1 + tx2
Pipelined Scheduling (I)

\[ f_s = 1 \text{ MHz} \]

\[ \tau_1 = 0.1 \mu s \]
\[ \tau_2 = 0.5 \mu s \]
\[ \tau_3 = 0.1 \mu s \]
\[ \tau_4 = 0.1 \mu s \]
\[ \tau_5 = 0.1 \mu s \]

\[ \text{Pipeline} \]
Block size: 100 samples
Time slot: \(100/f_s = 100 \mu s\)
Pipelining stages \(S\): 4
Processing latency: \(S \cdot T_{ts} = 400 \mu s\)

\[ \text{Processing latency} = 400 \mu s \]

**Pipelined execution:** removes precedence constraints, simplifies scheduling
**Pipelined Scheduling (II)**

2 processors of half capacity

![Diagram of pipelined scheduling](image)

- ADC
- $f_s = 1 \text{ MHz}$

**Equal latency and throughput**

- Block size: 100 samples
- Time slot: $100/f_s = 100 \mu s$
- Pipelining stages $S$: 4
- Processing latency: $S \cdot T_{ts} = 400 \mu s$

**Scheduling performance:** platform-independent

Processing latency = 400 µs
More powerful processors

ADC

$\tau_1 = 0.05 \, \mu s$

$\tau_2 = 0.25 \, \mu s$

$\tau_3 = 0.05 \, \mu s$

$\tau_4 = 0.05 \, \mu s$

$\tau_5 = 0.05 \, \mu s$

ADC

$100 \, \mu s$

$25 \, \mu s$

$25 \, \mu s$

$25 \, \mu s$

$5 \, \mu s$

Pipeline stages $S$: 4
Processing latency: $S \cdot T_{ts} = 400 \, \mu s$

Equal latency and throughput

Scheduling performance: platform-independent
Pipelined Scheduling: Latency Control (I)

More powerful processors

ADC

\[ f_s = 1 \text{ MHz} \]

\[ \tau_1 = 0.05 \mu s \]

\[ \tau_2 = 0.25 \mu s \]

\[ \tau_3 = 0.05 \mu s \]

\[ \tau_4 = 0.05 \mu s \]

\[ \tau_5 = 0.05 \mu s \]

Block size: 50 samples

Time slot: \( 50/f_s = 50 \mu s \)

Pipelining stages \( S \): 4

Processing latency: \( S \cdot T_{ts} = 200 \mu s \)

Scheduling performance: platform-independent

Processing latency = 200 \( \mu s \)
Pipelined Scheduling: Latency Control (II)

More powerful processors

ADC

\[ \tau_1 = 0.05 \mu s \]

\[ \tau_2 = 0.25 \mu s \]

\[ \tau_3 = 0.05 \mu s \]

\[ \tau_4 = 0.05 \mu s \]

\[ \tau_5 = 0.05 \mu s \]

ADC

\[ f_s = 1 \text{ MHz} \]

Block size: 50 samples

Time slot: 50/\( f_s \) = 50 \( \mu \)s

Pipelining stages \( S \): 3

Processing latency: \( S \cdot T_{ts} = 300 \mu s \)

Scheduling performance: platform-independent
Pipelined Scheduling: Control Flow

ADC

$\tau_1 = 0.2 \mu s$

$\tau_2 = 1 \mu s$

$\tau_3 = 0.2 \mu s$

$\tau_4 = 0.2 \mu s$

$\tau_5 = 0.2 \mu s$

$100 \mu s$

$100 \mu s$

$100 \mu s$

$100 \mu s$

$100 \mu s$

$f_s = 1 \text{ MHz}$

Scheduling performance: platform-independent

PIPETLINE

Block size: 100 samples
Period $T: \frac{1}{f_s} = 100 \mu s$
Pipelining stages $S: 4$
Processing latency: $S \cdot T = 400 \mu s$
Mapping

- Application and platform models
- Any mapping algorithm
- Two general-purpose algorithms:
  - $t_w$-mapping: $O(m \cdot n^{w+1})$
  - $g_w$-mapping: $O(m \cdot n^w)$
- Cost function

$$\text{Cost} = \frac{\text{processing requirement}}{\text{available processing power}} + \frac{\text{bandwidth requirement}}{\text{available bandwidth}}$$

balance processing load  minimize data flows

DEMO 1: Computing Resource Management

- LTE 1.4 MHz
- Time slot: 1 ms
- Sampling frequency: 1.92 MHz
- Each time slot, 1920 complex samples are sent to/ received from the USRP
- Receiver has 7 pipeline stages: 7 ms latency
Soft Real-Time: Execution Trace

![Graphs showing execution time and period traces over time slots.]

- Exec time 1 [μs]
- Period 1 [μs]
- Exec time 2 [μs]
- Period 2 [μs]
Soft Real-Time: Execution Trace
Hard Real-Time: Execution Trace

- Graphs showing execution time and period over time slots.
Hard Real-Time: Latency
Waveform Design and Deployment
• ALOE waveform
  ▫ Processing modules
  ▫ Connections
  ▫ Parameters

• Module
  ▫ Computing requirements
  ▫ Configuration parameters
Module Execution Flow

START
LOAD
Register

INIT

STATUS
Configuration Interface setup
Precompute coefficients

RUN

Real-time loop

STOP

STATUS

Free resources Unregister

EXIT

work()

initialize()
Waveform Design and Deployment

Development

Implementation of DSP algorithms

- CRC
- Turbo Coder
- Code Blk Segm.
- Code Blk Concat.
- Rate Matching

Deployment

Waveform creation and execution

- Parameters
- Execution time slot
- Pipelining stages
- …
Module Development

1. Model
2. Code
   ```
   #include <header.h>
   main() {
     ...
   }
   ```
3. Standalone test
4. Test

Deployment
- executable
- MEX-file
- shared library
Module Template

**initialize()**

```c
int initialize() {
    ...
    param_get_int_name("dft_len", &dft_len);
    dft_plan = compute_dft_plan(dft_len);
    return 0;
}
```

**work()**

```c
int work(void **inp, void **out) {
    complex_t *input = inp[0];
    complex_t *output = out[0];
    int nsamples = get_input_samples(0);
    param_get_int_name("runtime_param", &my_param);
    run_dft(dft_plan, input, output);
    for (int i=0; i<nsamples; i++) {
        ...
    }
    return 0;
}
```

**stop()**

```c
int stop() {
    destroy_dft_plan(dft_plan);
}
```
Stand-alone Execution: Debugging

Execution time: 7665 ns.
FINISHED
Type ctrl+c to exit
Warning: empty y range [1:1], adjusting to [0.99:1.01]
Standalone Execution: Profiling (e.g. Valgrind)
Verification

Model

MEX-file

Command Window

```matlab
>>
>>
>> mex aloefft.c ../debug_make/libfft.a /usr/lib/i386-linux-gnu/libfftw3f.a

Warning: You are using gcc version "4.7.2-2ubuntu1". The earliest gcc version supported
with mex is "4.1". The latest version tested for use with mex is "4.2".
To download a different version of gcc, visit http://gcc.gnu.org

>> x=rand(128,1)+i*rand(128,1);
>> y=aloefft(x,{{"dft_points",128}});
[info at file ..src/fft.c, line 60]: Using 128 DFT points
>> figure
>> plot(abs(y))
>> figure
>> plot(abs(fft(x)/sqrt(128)))
>>
```
DEMO 2: Module Development
Waveform Design and Deployment

Development

Implementation of DSP algorithms

- CRC
- Turbo Coder
- Code Blk Concat.
- Code Blk Segm.
- Rate Matching

Deployment

Waveform creation and execution

- Parameters
- Execution time slot
- Pipelining stages
- ...

Implementation of DSP algorithms

Waveform creation and execution
Waveform Definition and Testing

1a. Model

2s. Sub-models

3a. m-files
- mex-files

1b. Test

2a. Test

3b. Test

Integration in ALOE

Matlab/Octave

Module Development
# Application Description File (.app)

<table>
<thead>
<tr>
<th>Field</th>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modules</td>
<td>name</td>
<td>Unique ID</td>
</tr>
<tr>
<td></td>
<td>binary</td>
<td>name and rel. location of the binary</td>
</tr>
<tr>
<td></td>
<td>mopts</td>
<td>Processing requirements (for mapping)</td>
</tr>
<tr>
<td>Variables</td>
<td></td>
<td>Configuration parameter values</td>
</tr>
<tr>
<td>interfaces</td>
<td>{src=&lt;source&gt;; dest=&lt;destination&gt;}, {...}, ...</td>
<td>Connection of modules: src: output interface(s) of source module</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dest: input interface(s) of destination module</td>
</tr>
<tr>
<td></td>
<td>{&lt;m1&gt;,&lt;m2&gt;,...}, (...)</td>
<td>Executes modules listed in each pair of brackets in a single pipelining stage</td>
</tr>
</tbody>
</table>

**Examples**

```python
{src=<source>; dest=<destination>}, {...}, ...
```

- src: output interface(s) of source module
- dest: input interface(s) of destination module
Pipelining Stages

PDSCH - Tx

Source → TrBlk CRC → Code Blk (CB) Segm. → CB CRC → Coder → Rate Matching → Code Blk Concat. → Scrambling

Scrambling → Modulation Mapping → Resource Mapping → De-MUX → IFFT #1 → CP #1 → MUX → Channel

Waveform Deployment
Join Stages

join stages =
  (M1, M2, M3, M4, M5, M6, M7, M8),
  (M9, M23),
  ...
  (M22, M36)
);
Waveform Deployment Tools

1. Waveform description file
   • Collection of sub-waveform description files

2. Decouple Tx-Rx
   • Tx writes to file, Rx reads from file
   • Modify file with Matlab (add channel noise/distortion)

3. Debug mode
   • Logging service

4. Real-time execution:
   • UHD support
   • Execution statistics
Control Module (I)

Waveform Deployment

Stage 1

Stage 2

Stage 3

Stage 4

Stage 5

2 time-slots delay

No delay
Waveform Deployment

Control Module (II)

1) PBCH decodes BW

2) Set $f_s$

Stage 1  Stage 2  Stage 3  Stage 4  Stage 5
LTE DL Waveform

Stage 1
- Bit-level processing of all channels

Stage 2+3
- Symbol-level processing of all channels

Channel

Stage 1
- Synchronization

Stage 2+3
- CP detach. + FFT

Stage 4
- PBCH Rx

Stage 5
- PCFICH Rx

Stage 6
- PDCCH Rx

Stage 7
- PDSCH Rx
LTE DL Tx

Stage 1

- source
- pdsch_tx
- pcfich_tx
- pdcch_tx
- pbch_tx

Stage 2

- ctrl_mux
- demux_tx

Stage 3

- ctrl
- symbol_tx: IFFT+CP
- symbol_tx: mux

nof_input=4

nof_outputs=14

IFFT + CP
pdsch_tx

_input -> crc_tb -> coder -> rate matching -> scrambling -> modulator -> _output
LTE DL Rx

Stage 1: synchro

Stage 2: symbol_rx: de_output
symbol_rx: FFT + CP
demux
FFT + CP rem.

Stage 3: mux_rx
nof_inputs=14

Stage 4: resde_map_pbch

Stage 5: pbch_rx

Ctrl

resde_map_pcfich

Ctrl

pcfich_rx
PCFICH

CFI \rightarrow \text{CFI encoding} \rightarrow \text{Scrambling} \rightarrow \text{QPSK Mod} \rightarrow \text{Layer Map*} \rightarrow \text{Precoding*} \rightarrow \text{Res. Map} 

(a)

CFI' \leftarrow \text{CFI decoding} \leftarrow \text{Scrambling} \leftarrow \text{QPSK Hard Demod} \leftarrow \text{Layer Map*} \leftarrow \text{Precoding*} \leftarrow \text{Res. Map} 

(b)
BCH

24 bits every 40 ms

(a)

(b)

Subframe 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, ...

Subframe 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, ...

CRC attach → Scrambling of parity bits → Tail biting Conv. coding → Ctrl. Rate Match.

CRC detach → Scrambling of parity bits → Tail biting Conv. decoding → Rate De-match.

Scrambling → QPSK Mod → Layer Map* → Pre-coding*

Scrambling → QPSK Mod → Layer Map* → Pre-coding*

Scrambling → QPSK Mod → Layer Map* → Pre-coding*

Scrambling → QPSK Mod → Layer Map* → Pre-coding*

Res. Map

Res. Demap.
PDCCH

A1 bits

K1=A1+16 bits

3*K1 bits

E1 bits

An bits

Kn=An+16 bits

3*Kn bits

En bits

CRC 16 parity bits

Scrambling of parity bits if the case

Tail biting Conv. coding

Ctrl. Rate Match.

PDCCH MUX

Scrambling

QPSK Mod

Layer Map*

Pre-coding*

Res. Map

CRC 16 parity bits

Scrambling of parity bits if the case

Tail biting Conv. coding

Ctrl. Rate Match.

(b)
DEMO 3: Waveform Deployment
Conclusions
Conclusions

SDR Frameworks

- **SCA (Software Communication Architecture)**
  - Military
  - Research and education (e.g. OSSIE)

- **GNU Radio**
  - Research and education (PC, multicore)

- ...


ALOE Characteristics

Context

Heterogeneous Multiprocessing

Platform independence

portability

Development & deployment tools

Real time

Execution control

Resource management

Deterministic latency
Conclusions

multicores

many-cores

small clusters (heterogeneous)

DSP  GPP  FPGA

ALOE
Future Work

• GNU Radio/SCA compatibility

• Add and test new schedulers:
  ▫ Dynamic, provided by the RTOS
  ▫ Hybrid (static-dynamic)

• Tools
  ▫ Waveform development and deployment
  ▫ Graphical User Interface for ALOE++
  ▫ ...

Call for Participation

- **FlexNets** (Flexible Wireless Communications Systems & Networks)
  - http://flexnets.upc.edu/trac/
    - ALOE releases
    - Computing resource management framework
    - Waveforms
    - Educational material

- **OSLD** (Open source LTE deployment)
  - https://sites.google.com/site/osldproject/home
  - https://github.com/flexnets
    - ALOE++
    - DSP modules library
    - Development Tools

- **Mailing lists**: https://groups.google.com/group/flexnets