ALOE Session 5: Multiprocessing

Ismael Gomez, Vuk Marojevic, Antoni Gelonch
Universitat Politècnica de Catalunya

April 2011

1. Objective

The objective of this session is learning how to use ALOE on multiprocessor platforms. You will learn how to set up a multiprocessor environment for loading and running waveforms.

2. Overview

- Introduction to multiprocessing
- ALOE on multicore processors
- ALOE on multiprocessor platforms

3. Requirements

- Multicore processor (1st part), 2 PCs (2nd part)
- Linux, kernel 2.6.21 or above
- ALOE version 1.3 downloaded and installed on both PCs (see ALOE Session 1)
- Basic Linux user skills

4. ALOE versions

We continuously evolve the ALOE framework and tools. This session is compatible with ALOE version 1.3. Consult http://flexnets.upc.edu/trac/wiki/ALOEedu for updates.

5. Procedure

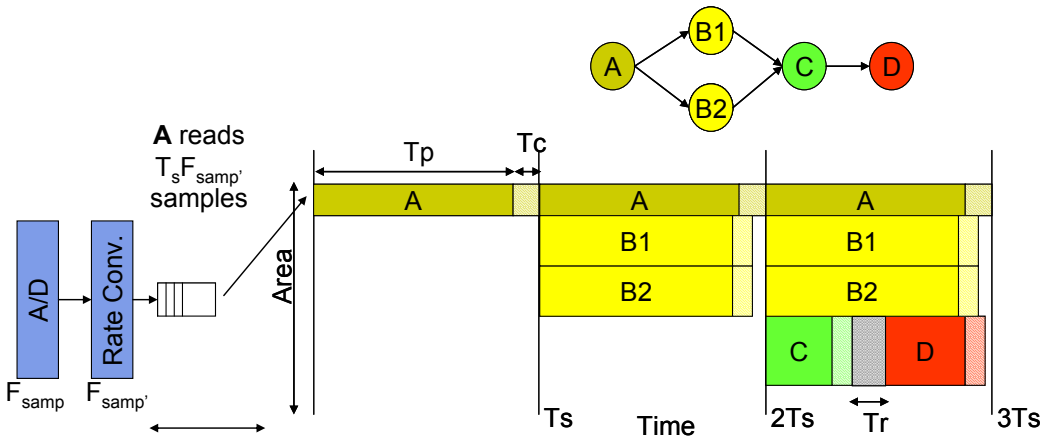
5.1. Introduction to multiprocessing

Signal processing tasks for wireless communications systems are algorithmically complex, in general. These tasks often need to be executed on power-constrained computing platforms. Hence, computing efficiency is an important topic for the telecommunications system designer and operator.

The signal processing becomes more complex with the introduction of new wireless communications standards. Higher transmission bandwidths and sampling rates also have implications on the computing complexity. Fortunately, signal processing algorithms are parallelizable. These algorithms consist of a set of mathematical operations that are applied on a continuous data flow. Hence, multiple processors can perform operations in parallel, increasing the processing throughput, or decreasing the processor frequency and power consumption, at the cost of a higher processing latency. This technique, called *pipelining*, is common in digital system design. Figure 1 illustrates the concept.

A pipelined execution eliminates the data flow dependencies between tasks. This enables executing tasks concurrently. Tasks A, B1, and B2 of Figure 1, for example, can then run on three processors. A and B1 or B2, however, process different data chunks in a given time slot. The example of Figure 1 assumes 4 available processors so that tasks C and D are executed sequentially. Pipelining increases the task execution periodicity and, thus, the data throughput.

In case of an SDR receiver, the data processing needs to be synchronized with the analog-to-digital converter (A/D) that continuously feeds the digital signal processing chain with new data samples. The transmitter works analogous, but synchronizes the task execution periods with the digital-to-analog converter (D/A). This kind of synchronization is necessary on all processors. Otherwise, data samples may be lost or excessive buffering required.



Clocks are synchronized periodically in frequency, not phase.
This enables having multiple A/D from multiple antenna with different phases.

Figure 1 - Pipelined execution pattern.

The abstraction layer and operating environment (ALOE) is a middleware for SDR applications, or waveforms. ALOE enables platform-independent design, where the waveform description is isolated from the platform architecture. It supports multiprocessing platforms and features an automatic computing resource manager that maps application modules (implementing basic or advanced digital signal processing tasks) to distributed computing resources. The time management ensures a correct scheduling and synchronization of data flows, supporting a pipelined execution.

5.2. Configuring ALOE for a multicore processor

We first examine symmetric multiprocessing (SMP) multicore processors. A multicore processor is a single chip featuring multiple processing cores. The individual cores share several peripherals—memories, accelerators, interfaces, cache, and so on—enabling the communication of threads running on different cores. Multicore processors are symmetric in the sense that the time needed to perform a task is exactly the same on each core, simplifying the task scheduling process. This section describes how to configure ALOE to run on a multicore processor.

We assume that ALOE has already been installed on your multicore processor. Otherwise, perform the installation steps described in ALOE Session 1.

A single parameter controls the number of cores that ALOE will consider. Edit the */usr/local/etc/platform.conf* file with root permissions and set the variable **nof_cores** to the number of cores on your platform. Choose 2 if you are using a Dual Core PC.

Now you can start ALOE and load, initialize, and run waveforms as usual (see ALOE Session 1). If you choose loading the ***example*** waveform, you will notice that it maps to a single core. This is so because of its low processing load and simple data flow dependencies. If you try loading the ***utran*** waveform, on the other hand, you should observe how the modules are distributed between the two cores. This waveform requires a significant amount of processing resources and features modules with no data flow dependencies.

5.3. Configuring ALOE for a multiprocessor platform

The second scenario assumes individual processors, which do not share memory or any other peripheral, but, rather, communicate through an external data interface, such as an Ethernet link.

ALOE currently supports TCP and UDP over Ethernet. Both protocols lack quality of service (QoS) mechanisms. Interprocessor data flows may, therefore, lead to real-time processing violations. Make sure that the network load is low for minimizing this effect.

You will configure a simple network of two processors, PC1 and PC2. PC1 will execute the ALOE Manager Daemons as well as some waveform modules, whereas PC2 will execute the remaining waveform modules. The network consists on only one

unidirectional interface, which connects PC2 to PC1. Another interface is used for synchronization purposes. PC1 will provide the time reference.

ALOE needs to be installed on both PCs, connected through an Ethernet interface. Make sure that you have TCP/IP connectivity between the two processors. You can use *ping* for this purpose,

```
ping 192.168.2.2
```

replacing `192.168.2.2` with your processor's IP address.

You can now configure the external interfaces of PC1. Therefore, edit */usr/local/etc/xitf.conf* as root on PC1 to match Figure 2.

Figure 2 - *xitf.conf* file of PC1.

```
# slave control itf
[xitf]
      id=0x10
      address=0.0.0.0
      port=7000
      mode=inout

# input data itf
[xitf]
      id=0x80
      address=0.0.0.0
      port=9000
      mode=in

# sync master input itf
[xitf]
      id=0x3
      address=0.0.0.0
      port=8000
      mode=inout
```

The data interface is an input interfaces; the IP address 0.0.0.0 or any local address can be specified. The mode *inout* (input/output) indicates a bidirectional interface that will be used for control and synchronization purposes. The local processor creates and initiates the socket. Again, you can use 0.0.0.0 or any other local address. The *outin* (output/input) interface is also bidirectional, but the local processor connects to a remote socket instead of initiating it.

The interface IDs indicate:

- 0x2: synchronization slave (*sync*) interface (*outin* mode only),
- 0x3: synchronization master (*sync_master*) interface (*inout* mode only),
- 0x1: master control interface,

- 0x1n, n=[0..F]: slave control interface,
- 0xpq, p=[2..F],q=[0..F]: data interfaces.

Now edit the */usr/local/etc/xitf.conf* file of PC2. Configure it to connect to the IP address and ports of PC1 (Figure 3).

Figure 3 - *xitf.conf* file of PC2.

```
# master control itf
[xitf]
      id=0x1
      address=192.168.2.1
      port=7000
      mode=outin

# output data itf
[xitf]
      id=0x80
      address=192.168.2.1
      port=9000
      mode=out

# sync slave output itf
[xitf]
      id=0x2
      address=192.168.2.1
      port=8000
      mode=outin
```

Note that you need to substitute IP address **192.168.2.1** with the IP address of PC1. To obtain this address you can use the `ifconfig` command with your physical interface as an argument, for example:

```
ifconfig eth0
```

We will configure the set of daemons that will execute on the two processors next. Open the */usr/local/etc/platform.conf* file with root permissions on PC1 and edit the *daemons* variable to feature the following 10 daemons:

```
daemons=cmdman,swman,hwman,statsman,frontend,exec,swload,stats,bridge,
sync_master
```

This will launch five manager daemons *cmdman, swman, hwman, statsman*, and *sync_master* and five sensor/actuator daemons *frontend*, *exec*, *swload*, *stats*, and *bridge* on PC1.

On PC2 we will launch the mandatory daemons only. Therefore, open */usr/local/etc/platform.conf* on PC2 and change the *daemons* line to

```
daemons=frontend,exec,swload,stats,bridge,sync
```

Note that the *sync_master* daemon will be launched on PC1, whereas *sync* will run on PC2. This means that PC1 provides the time reference for PC2.

ALOE currently supports only one slave for each synchronization master. Therefore, in a network of three or more processors, the *sync_master* and *sync* daemons need to be placed in a cascade. For a network of three processors, for example, P1 may execute the *sync_master* for P2, P2 the corresponding *sync* (slave) and *sync_master* for P3, which runs the *sync* (slave).

Since we are using Linux sockets, it is important to take care of the order in which ALOE is initiated and stopped on the different processors. We recommend launching ALOE on PC1 first, because PC1 creates a socket that will be used by PC2. When finishing, terminate the ALOE processes in the reverse order. This way sockets are cleanly initiated and terminated without the need for wait states.

After launching ALOE on both processors, you should observe the messages of Figure 4 appearing on the shell of PC1.

Figure 4 - Messages on PC1 after launching ALOE on both processors.

```
…
HWMAN: Add processor: New CPU: PE 0x2, 2000000 MACS 1 cores TS=10000
us
…
HWMAN: Add processor: New CPU: PE 0x4, 2000000 MACS 1 cores TS=10000
us
```

These massages indicate that the hardware manager (*hwman* or HWMAN) has detected both processors (PC1, or PE 0x2, and PC2, or PE 0x4). You will also see the processing capacity vector and the interprocessor bandwidth matrix.

Now you are ready to load a waveform. Notice that the command shell is accessible only at PC1, which runs the command manager (*cmdman*) daemon as specified in */usr/local/etc/platform.conf*.

Try loading and executing the ***utran*** waveform on your own (see ALOE Session 1, if necessary). Use the *execinfo* tool to check the distribution of waveform components between the processors or the aloeUI tools (ALOE Session 4) for visualizing the mapping and scheduling results.

This finishes ALOE Session 5. Please send us your feedback to flexnets.pmt@upc.edu.