ALOE Session 2: Creating a Waveform

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

April 2011

1. Objective

In this session you will experience some basic ALOE functionalities. You will learn how to create and run a custom waveform consisting of a set of predefined modules.

2. Overview

- Application Description File (.app) terminology
- Module interfaces
- Create Application Description File and set initialization parameters
- Load and run a new waveform

3. Requirements

- PC running Linux, kernel 2.6.21 or above
- ALOE version 1.3 downloaded and installed on your PC (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. The Application Description File

Directed graphs can characterize the data processing and data flow characteristics of waveforms, where nodes represent data processing and arcs data flows. Figure 1 shows the ***example*** waveform. Each block performs some data processing. Processed data is propagated to the next block in the processing chain as indicated by the arrows. We use *module*, *component*, and *object* interchangeably for referring to a digital signal processing block.

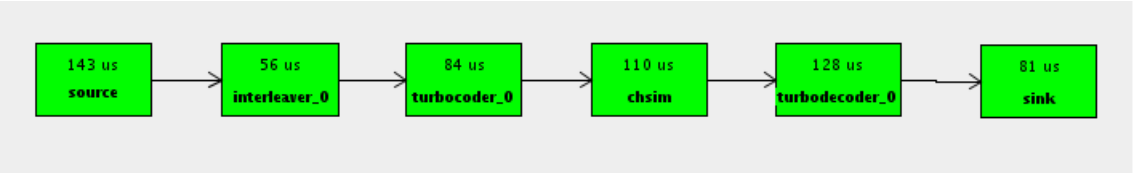

Figure 1 - Graphical representation of the *example* waveform

A simple text file can specify such a graph. This text file is located in the *swman_apps/* folder and is identified by the *.app* extension: *WaveformName.app*.

Go to the directory where you extracted and installed the ALOE source package on your PC. We will refer to it as *$ALOE/* (*aloe-<version_number>/* if you followed the installation instructions of ALOE Session 1). Now switch to *example-repository/swman_apps/* and you will find *example.app* (Figure 8, appendix). This file provides the following information about each module:

- **obj_name:** name used by ALOE to refer to this module,
- **exe_name:** name of the executable file,
- **proc:** processing requirements.

It also defines the modules' input and output interfaces (inputs and outputs):

- **name:** name of the interface,
- **remote_itf:** name of the remote interface (at the other end of the link),
- **remote_obj:** name of the remote object (the module at the other end of the link),
- **bw:** bandwidth requirement of the link.

5.2. Selecting components

We first need to select the components that will assemble the desired waveform. We exemplify this using a simple BPSK modulator. Go back to *$ALOE/*. The *modules/* directory contains several sub-directories, one for each available component:

```
modules/
            gen_binsource/
            gen_cconv/
            gen_channel_gauss/
            gen_chmux/
            gen_crc/
            …
```

The prefix *gen_* indicates that the component does not comply with a specific radio standard. Each component directory has the same structure, as illustrated below for the ***gen_digitalmodem*** component.

```
gen_digitalmodem/
                  doc/              Component documentation
                  interfaces/       Interface definitions
                  lnx_make/         Compiler files for Linux
                  src/              Source code
```

The *interfaces/* directory contains the following files:

- *inputs.h*: input interfaces description,
- *outputs.h*: output interfaces description,
- *stats.h*: statistics and initialization parameters description.

We examine the input and output interface descriptions first. Figure 2 shows the contents of the *input.h* file of the ***gen_digitalmodem*** component.

Figure 2 - Input interfaces description (*input.h*) of the *gen_digitalmodem* component.

```
/** Input is bitstream for modulator, or 2 arrays of shorts (I & Q)
for demodulator */

#define INPUT_MAX_DATA  100*1024    /*Max Number of data symbols*/

typedef char input1_t;
input1_t input_data[INPUT_MAX_DATA];
int process_input(int len);


/** configure input interfaces */
struct utils_itf input_itfs[] = {
                    {"input",             /* interface name */
                    sizeof(input1_t),     /* sample size */
                    INPUT_MAX_DATA,       /* input buffer size */
                    input_data,           /* input buffer */
                    NULL,
                    process_input},       /* processing function */

                    {NULL, 0, 0, 0, 0, 0}};
```

This file defines the *input_itfs* structure, containing the following information:

- **Interface name:** "input",
- **Sample size:** bitstream packet in bytes (type **char**) if modulating, 16-bit signed integers (type **short**) if demodulating,
- **Input buffer:** array that stores the data,
- **Buffer size:** maximum buffer size,
- **Processing function:** function that will be called when data is available for processing.

The ***gen_digitalmodem*** component can operate as a modulator or as a demodulator; this is the reason for defining different data types for the *sample size*. Later we will discuss how to select the module's operating mode.

To connect two modules through a data link, we need the following interface information:

1) Output interface name and data type of the sender module
2) Input interface name and data type of the receiver module

Building a waveform is then as simple as choosing the signal processing components, identifying the interfaces, ensuring data type compatibility, and generating the Application Description File (.app).

5.3. Creating the Application Description File

We create the Application Description File using the following components:

1. gen_binsource
2. gen_digitalmodem (configured as a modulator)
3. gen_channel_gauss
4. gen_digitalmodem (configured as a demodulator)
5. gen_sink

Switch to *$ALOE/example-repository/swman_apps/* and open *modem.app* in a text editor. Modify this file to match that of Figure 9 (appendix). Then save it.

5.4. Module parameters and statistics

Some waveform components are configurable. One or a set of module-specific parameters can specify the module's functionality in the given processing chain (waveform). This configuration is done offline, that is, before the real-time data processing starts. Hence, there are no timing restrictions during the waveform initialization phase.

Some modules may have predefined parameter values; the parameter configuration is optional for these. The set of module-specific parameters is described in the *stats.h* file under the module's *interfaces/* directory.

Figure 3 depicts the contents of the *stats.h* file of ***gen_digitalmodem*** (*$ALOE/modules/gen_digitalmodem/interfaces/*). It contains two parameters:

- *mod_demod* indicates if the component will acts as a modulator or a demodulator,
- *modulation* configures the modulation type: BPSK, QPSK, ….

The first structure specifies these parameters, containing the following data fields:

- **Parameter name**: name used by the user to set the value,
- **Data type:** data type, which can be one of the following:
  - STAT_TYPE_CHAR/STAT_TYPE_UCHAR: 1 byte signed/unsigned integer,
  - STAT_TYPE_SHORT/ STAT_TYPE_USHORT: 2 bytes signed/unsigned integer,
  - STAT_TYPE_INT/ STAT_TYPE_UINT: 4 bytes signed/unsigned integer,
  - STAT_TYPE_FLOAT: 4 bytes floating point representation,
  - STAT_TYPE_STRING: string of characters,
- **Data size:** maximum number of array elements of the specified type,
- **Data pointer:** pointer to the variable that stores the parameter value.

The parameter set of a component that is reused in several waveforms differs from waveform to waveform. Therefore, a file specifies the parameters for each component and waveform. This file can be found under the component directory of the waveform. The name of the file is the name of the **object** indicated in the *.app* file, but with the *.params* extension. (Note that a module that is deployed in different waveforms has the same object name, although the names of the corresponding executables may differ).

The *stats.h* file also defines a set of variables that can be modified and viewed during waveform execution. These variables are called **statistics**. Keep in mind that modifying a statistics, as opposed to a parameter, may lead to timing violations. Increasing the maximum number of turbo decoder iterations, for example, will boost the module's computing complexity. The component may then exceed the available computing resources, leading to real-time violations.

The second structure in *stats.h* (Figure 3) defines the statistics. It contains the following fields:

- **Stat name**: name used by the user to modify or visualize the variable,
- **Data type:** data type, same options as above,
- **Data size:** maximum number of array elements of the specified type,
- **Stat handler pointer:** used to manually set or get variable values through the ALOE software API (*SWAPI*),
- **Data pointer:** pointer to the statistics value or values,

Figure 3 - Parameters and statistics file (*stats.h*).

```
/** Stats & parameters files declarations */

/** statistic Id's variables */
int stat_modulation,stat_outsignal_i,stat_outsignal_q;

/** parameter variables */
int mode,modulation;

output1_t output_data_i[100];
output1_t output_data_q[100];

struct utils_param params[] = {
    /* 0 modulator
     * 1 demodulator
     */
    {"mod_demod",       /** parameter name */
      STAT_TYPE_INT,    /** data type */
      1,                /** data size */
      &mode},           /** data pointer */

    /* 0 bpsk
     * 1 qpsk
     * 2 qam4
     * 3 psk8
     * 4 qam16
     */
    {"modulation",STAT_TYPE_INT,1,&modulation},
                            {NULL, 0, 0, 0}};


struct utils_stat stats[] = {
                             {"modulation",     /** stat name */
                            STAT_TYPE_INT,      /** data type */
                            1,                  /** data size */
                            &stat_modulation,   /** stat handler ptr */
                            (void*)&modulation,/** data ptr */
                            READ},              /** r/w mode */
/** supported modes are:
      - READ: Obtains the value before calling processing functions
      - WRITE: Sets the value after calling processing functions
      - OFF: manually use SetStatsValue/GetStatsValue to read/write
*/
                             {"out_signal_i",
                            STAT_TYPE_CHAR,
                            100,
                            &stat_outsignal_i,
                            (void*)output_data_i,
                            WRITE},

                             {"out_signal_q",
                            STAT_TYPE_CHAR,
                            100,
                            &stat_outsignal_q,
                            (void*)output_data_q,
                            WRITE},

                            {NULL, 0, 0, 0, 0, 0}};
```

- **R/W mode:** allows the user to automatically read or write the contents of the variable. The supported modes are:
  - *READ*: Obtains the value before the module's execution,
  - *WRITE*: Sets the value after the module's execution,
  - *OFF*: manually use the *SWAPI* functions to retrieve or set the value of a variable.

Below we summarize the parameter and statistics configuration options.

Parameters:

- Different for each waveform,
  - The structure `utils_param params[]` of the *interfaces/stats.h* file under the component directory specifies the parameters.
  - The parameter values are defined in *ObjectName.params*, where ObjectName is the name of the module due to the Application Description File. *ObjectName.params* is located in the *statsman/WaveformName/* directory.

Statistics:

- Waveform-independent,
- The structure `utils_stat stats[]` of the module's *interfaces/stats.h* filespecifies the statistics.

5.5. Setting module parameters

We now define the parameters of our ***modem*** waveform. Switch to *$ALOE/example-repository/statsman/modem/*. Create or modify the four files *source.params*, *modulator.params*, *channel.params*, and *demodulator.params* to match with Figures 4 to 7.

Figure 4 - *source.params*

```
parameter {
      name = numbits
      value = 1024
}
```

numbits specifies the number of bits that the ***source*** module generates each time slot.

Figure 5 - *modulator.params*

```
parameter {
      name = mod_demod
      value = 0
}
parameter {
      name = modulation
      value = 0
}
```

mod_demod = 0 and *modulation* = 0 configures ***gen_digitalmodem*** to operate in the BPSK modulator mode (see *$ALOE/modules/gen_digitalmodem/interfaces/stats.h*).

Figure 6 - *channel.params*

```
parameter {
      name = output_data_type
      value = 1
}

parameter {
      name = gain
      value = 80.0
}
```

output_data_type = 1 defines the data type of the output signal as an 8-bit signed integer (see *$ALOE/modules/gen_channel_gauss/interfaces/stats.h*), whereas *gain* stands for the amplification factor.

Figure 7 - *demodulator.params*

```
parameter {
      name = mod_demod
      value = 1
}
parameter {
      name = modulation
      value = 0
}
```

mod_demod = 1 and *modulation* = 0 indicates the BPSK demodulator mode (see *$ALOE/modules/gen_digitalmodem/interfaces/stats.h*).

The ***sink*** component does not need any parameters.

5.6. Loading the new waveform

Everything is set now for loading and executing the new waveform. Switch to the *$ALOE/* directory. Then launch ALOE and load, init, and run the ***modem*** waveform:

```
sudo runph
runph$: phload modem
runph$: phinit modem
runph$: phrun modem
```

The waveform should be running. Pause the waveform execution and use the *execinfo* tool to obtain the components' execution statistics:

```
runph$: phpause modem
runph$: execinfo modem
```

Remember that if continuous messages appear—indicating real-time violations—you can still type `phpause modem` and press enter to pause the waveform execution.

Increase the data throughput of the waveform by setting the *numbits* variable of the ***source*** component to 2048, for example:

```
runph$: statset modem source numbits 2048
```

Observe the consumed resources now.

Finally, try changing the modulation type. Set the *modulation* variable to 1 to switch to QPSK:

```
runph$: statset modem modulator modulation 1
```

With this we finish ALOE session 2. Please send your feedback to flexnets.pmt@upc.edu.

Appendix

Figure 8 - Application Description File of the *example* waveform (*example.app*)

```
object {
      obj_name=source
      exe_name=gen_binsource
      proc=47
      outputs {
            name=output
            remote_itf=input
            remote_obj=interleaver_0
            bw=1000
      }
}

object {
      obj_name=interleaver_0
      exe_name=gen_interleaver
      proc=47
      inputs {
            name=input
            remote_itf=output
            remote_obj=source
            bw=1000
      }
      outputs {
            name=output
            remote_itf=input
            remote_obj=turbocoder_0
            bw=1000
      }
}

object {
      obj_name=turbocoder_0
      exe_name=utran_turbocoder
      proc=60
      inputs {
            name=input
            remote_itf=output
            remote_obj=interleaver_0
            bw=1000
      }
      outputs {
            name=output
            remote_itf=input
            remote_obj=chsim
            bw=1000
      }
}

object {
      obj_name=chsim
      exe_name=gen_channel_gauss
      proc=572
      inputs {
            name=input
            remote_itf=output
```

```
            remote_obj=turbocoder_0
            bw=1000
      }
      outputs {
            name=output
            remote_itf=input
            remote_obj=turbodecoder_0
            bw=1000
      }
}

object {
      obj_name=turbodecoder_0
      exe_name=utran_turbodecoder
      proc=358

      inputs {
            name=input
            remote_itf=output
            remote_obj=chsim
            bw=1000
      }
      outputs {
            name=output
            remote_itf=input
            remote_obj=sink
            bw=1000
      }
}

object {
      obj_name=sink
      exe_name=gen_sink
      proc=47
      inputs {
            name=input
            remote_itf=output
            remote_obj=turbodecoder_0
            bw=1000
      }
}
```

Figure 9 - *modem.app*.

```
object {
	obj_name=source
	exe_name=gen_binsource
	proc=47
	outputs {
		name=output
		remote_itf=input
		remote_obj=modulator
		bw=1000
	}
}

object {
	obj_name=modulator
	exe_name=gen_digitalmodem
	proc=47
	inputs {
		name=input
		remote_itf=output
		remote_obj=source
		bw=1000
	}
	outputs {
		name=output
		remote_itf=input
		remote_obj=channel
		bw=1000
	}
}

object {
	obj_name=channel
	exe_name=gen_channel_gauss
	proc=572
	inputs {
		name=input
		remote_itf=output
		remote_obj=modulator
		bw=1000
	}
	outputs {
		name=output
		remote_itf=input
		remote_obj=demodulator
		bw=1000
	}
}

object {
	obj_name=demodulator
	exe_name=gen_digitalmodem
	proc=358

	inputs {
		name=input
		remote_itf=output
		remote_obj=channel
		bw=1000
	}
```

```
      outputs {
            name=output
            remote_itf=input
            remote_obj=sink
            bw=1000
      }
}

object {
      obj_name=sink
      exe_name=gen_sink
      proc=47
      inputs {
            name=input
            remote_itf=output
            remote_obj=demodulator
            bw=1000
      }
}
```