Project no: FP6-2002-IST-C 020023-2

Project title: FlexCode

Instrument: STREP

Thematic Priority: Information Society Technologies

D4.3 Report on FlexCode Hardware Demonstration

Due date of deliverable: 2009-06-30
Actual submission date: 2008-08-15

Start date of project: 2006-07-01 Duration: 36 Months

Organisation name of lead contractor for this deliverable: Nokia

Revision: 0.4

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

1 Summary

This report describes the integration of the *FlexCode* codec and channel model in a real-time hardware demonstrator. The demonstrator is a voice over IP (VoIP) type of real-time application enabling the assessment of the flexible operation of the *FlexCode* codec in heterogeneous network environments with different speech and audio content. The demonstrator enables both real-time conversation and audio streaming experiments.

The demonstration hardware consists of two laptop PCs connected to each other with fixed LAN network, WLAN or ad-hoc WLAN configuration. The actual IP connection between *FlexCode* clients is considered transparent regarding the channel conditions so that demonstrated network conditions are totally controlled with the *FlexCode* channel model included in the demonstration platform.

The demonstration was implemented using open source software modules, tools and protocols. The PC clients have a Linux operating system while the VoIP protocol stack, data transmission and the real-time media handling including audio capture and representation was implemented using the GStreamer media framework. The demonstration application and graphical user interface was built using the Maemo platform and SDK on Linux.

2 Introduction

Work package 4 (WP4) of *FlexCode* Project was responsible for integrating the source and channel codec implementations originating from work packages 1 and 2 into a single real-time demonstration environment. In addition, the *FlexCode* channel model was included in the framework to demonstrate the codec operation in heterogeneous network conditions.

An open source environment was selected for the demonstration framework. The environment consists of a Linux operation system, the Maemo platform and SDK for the demonstration application and graphical user interface development and GStreamer for real-time media handling, audio capture and representation, and transport protocols for VoIP clients. The *FlexCode* codec including the channel model simulation was integrated as a GStreamer element plug-in in the media framework.

This report covers first the integration of the *FlexCode* codec algorithms, tools and simulations within the hardware demonstration. Integration of the *FlexCode* codec and channel model interface including the adaptation and control parameters and required control data transmission mechanisms of the demonstration are described in Section 3. The GStreamer framework applied in the demonstration as well as the *FlexCode* integration into it is reviewed in Section 4. The demonstration platform graphical user interface, software and hardware architecture are described in Section 5. Finally, the conclusion is provided in Section 6.

3 Codec integration

Since the goal of the real-time demonstration is to demonstrate the *FlexCode* codec functionality and enable codec assessment regarding the adaptation capability and flexibility, the codec algorithm implementation needs to run in real-time on the selected platform and have extensive user control and adaptation possibilities. Normally, a real-time conversational application or service would hide both the adaptation and complicated operation-mode selections from the user, but in this case, all the functionalities need to be available for the user.

3.1 FlexCode source and channel codec

The *FlexCode* codec consisting of a source and a channel codec as well as a channel model was received from work packages 1 and 2 in C/C++ format. The channel coder and channel simulation was originally created using C++, but the source coding part needed to converted to C implementation particularly for the real-time demonstration together with WP1. The implementation was split into separate encoding and decoding modules for the real-time demonstration framework. Independent encoder and decoder implementations were needed especially for the full-duplex conversational application.

3.2 Codec control and adaptation

The hardware demonstrator consists of two VoIP clients. For the demonstrator user the system appears as local- and far-end client. For demonstration purposes, and to assess the real-time system performance, the user needs to be able to control and change the source and channel encoder functionality in the local- and far-end client. For example, if the user wants to evaluate the effect of source bit rate, the encoder in the far-end client needs to be controlled with request signalling. That is, some of the user control signals of the codec and channel model are not only for the local client, but need to be forwarded to the other client. Therefore, the demonstration framework needed a special control plane for handling the *FlexCode* codec and channel model adaptation and control signalling.

Table 1 lists the designed user controls and signalling for the *FlexCode*. Naturally, the local encoder and decoder control signalling does not require any transport of the user command since the control affects the implementation in the local client. For example, the channel conditions, such as packet loss rate and ES/N0, are parameters that affect only the channel model and decoder implemented in the local client. On the other hand, some of the control signals need to be transmitted to the far-end client. For example, the design packet-loss rate and the design ES/N0 are parameters that affect the tuning of the source and channel encoder locating in the far-end client.

Table 1 User control parameters and signalling.

Control parameter	**Value range**	**Target**	**Location**
Coding mode I	KLT/MDCT	Encoder and decoder	Local
Encoder bit rate (command)	12 – 64 kbit/s	Encoder	Local
Encoder bit rate (request)	12 – 64 kbit/s	Encoder	Far end
Coding mode II (command)	MDC/bit-plane	Encoder	Local
Coding mode II (request)	MDC/bit-plane	Encoder	Far end
Design packet loss rate	0 – 50%	Encoder	Far end
Actual packet loss rate	0 – 50%	Channel model (decoder)	Local
Design ES/N0 (request)	-10 – 40 dB	Encoder	Far end
Actual ES/N0	-10 – 40 dB	Channel model (decoder)	Local
Bit-plane control	0 – 50 bits	Decoder	Local

The encoder bit rate command affects the source coding rate at the local client. The bit rate request is transmitted to the far-end client and changes the bit rate the local client is receiving. The functionality enables the user to assess the source coding performance at different bit rates.

The design packet loss rate request affects the *FlexCode* multiple description coding (MDC) approach in the far-end source encoder. The purpose of this functionality is to assess the flexibility of the codec in case the receiver is expected (or is measured) to experience various packet loss rate conditions. The source and channel encoder may decide to change MDC strategy based on the given loss rate. The design ES/N0 control request works similarly. Based on the estimated channel conditions in the receiver, the far-end channel encoder may adapt the strength or scheme-used of the forward error correction.

The actual packet loss rate and the actual ES/N0 parameters affect the local channel model. Based on the user command the channel model simulator creates the desired channel condition.

MDC and bit-plane coding are mutually-exclusive tools in *FlexCode*. That is, the bit-plane coding is available only when single description coding (SDC) is used. In practice, if the user selects bit-plane coding, the far-end encoder and local decoder are forced to SDC mode. The bit-plane coding enables fine granularity bit rate control. Although the bit rate reductions would normally be done already at the far end before transmission, the simulator implementation has simplified the approach and simulates the functionality in the receiver side. The perceived effect is naturally the same.

Being able to set both design and actual channel error condition parameters in the simulation, the user may assess the *FlexCode* codec performance in case the adaptation of the channel and source coding is aligned with the prevailing channel conditions and when there is a possible mismatch in the codec functionality.

All the parameters listed in Table 1 can be tuned during the simulation even in every frame. Hence, the user has a good possibility to assess the codec performance. The only exception is the selection between the KLT and MDCT coding models. Due to the different approach in the source coding, this functionality cannot be changed during a simulation but needs to be selected in the beginning.

3.3 In-band signalling

The user control signals, as described above in Table 1, can be classified into two categories: 1) local codec and 2) far-end codec control parameters. The codec implementation has an application interface for both category signals, but, as discussed above, for far end control parameters a special control plane was needed as well. *FlexCode* demonstration was designed to carry the control signalling within the codec bit stream as in-band signalling.

In-band signalling approach is straightforward and flexible to implement. Signalling is obviously very fast and always synchronised with the actual media since it is carried within the data frames together with the coded bits. The design assumption of the demonstration was that the simulated control plane signalling was well protected, and hence, error free. Therefore, the control parameters carried within the data frames were not affected by the simulated channel conditions.

4 Demonstration framework

Media handling, as well as media and data transport between *FlexCode* demonstration clients, was realised with the open source GStreamer framework also available as an open source project in http://gstreamer.freedesktop.org. The advantage of GStreamer is the modular structure and readily available elements for real-time data transport and tools for audiovisual media capture and representation. The demo specific algorithms and tools, such as audio and channel coding and media rendering can be easily integrated within the framework.

4.1 GStreamer media framework

Figure 1 presents the basic media handling in modular architecture of the GStreamer platform. A basic media streaming or playback is realised with a pipeline consisting of a source element, a filter element, and a sink element. A source element can be, e.g., an audio file on a hard disk. In a conversational application like the *FlexCode* demonstration, the source element provides real-time audio capture with the selected sampling rate and data format. The filter element may consist of an encoder, a decoder or any other filtering functionality. Finally, the sink element can be an audio device that plays back the processed audio content or a file to store the processed media.

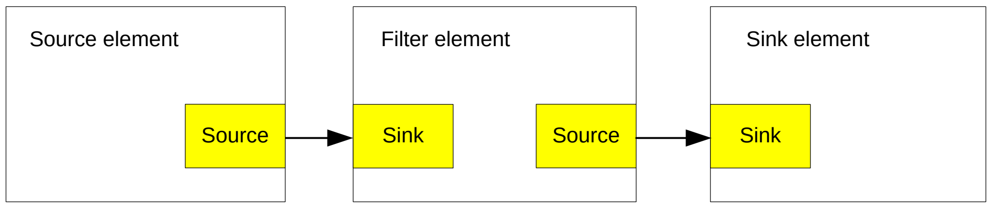

Figure 1 Basic GStreamer media handling framework consisting of modular elements.

The GStreamer elements also facilitate a data and a status transport between the pipeline and the application managing the pipeline. Pipeline status, such as the end of the streamed media file could be conveyed to the application level using the GStreamer Bus. In addition, the elements may have input and output arguments containing control and adaptation data. This feature can be applied in the application design for user control signalling.

4.2 FlexCode in the GStreamer framework

The *FlexCode* source and channel codec including the channel model was implemented as a GStreamer plug-in element. Due to the full-duplex application needs, the plug-in was split into independent encoder and decoder elements. The channel model was integrated into the decoder element together with the channel decoder.

The *FlexCode* plug-in was designed as a wrapper around the *FlexCode* implementation creating the sink and source interfaces towards GStreamer framework and application level user control. The element itself utilizes the interfaces provided by the *FlexCode* source and channel coding algorithms, tools and channel model implementation.

4.2.1 Source and channel encoder

Figure 2 presents the overview of the *FlexCode* encoder element consisting of source and channel encoding algoritms and in-band data transmission. The input to the element is 16 kHz mono audio signal in 16 bit linear format. The output consists of coded bits of the source and channel encoder and in-band signalling for controlling the far-end client.

The application level control parameters form two categories. As presented in Table 1 in Section 3.2, some of the parameters are for the local encoder, while other commands need to be transmitted to the far-end client.

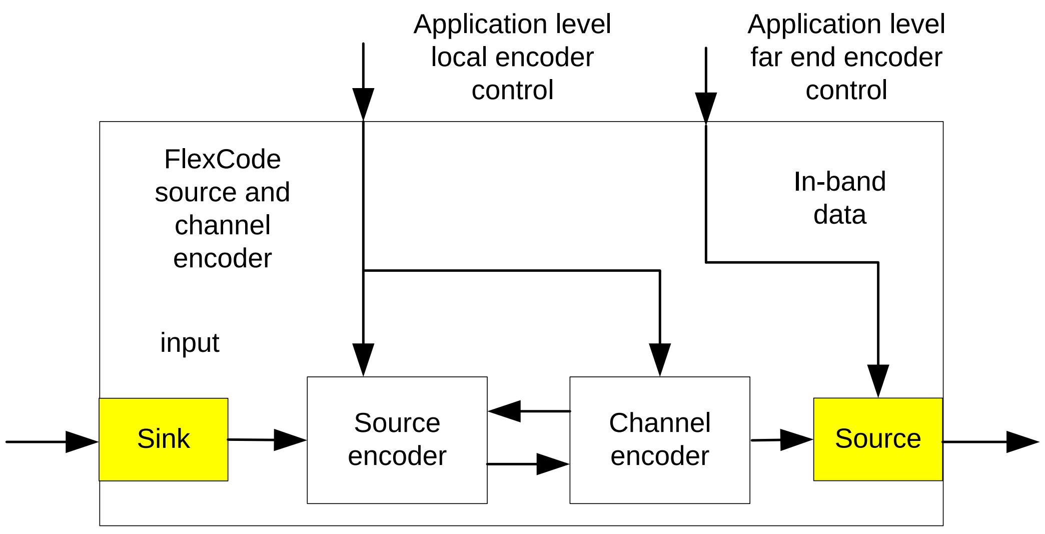
Application level local encoder control
Application level far end encoder control
FlexCode source and channel encoder
In-band data
input
Sink
Source encoder
Channel encoder
Source

Figure 2 *FlexCode* encoder element with application level control for local and remote source and channel encoder.

The GStreamer encoder implementation has several input arguments. The interface details on the input arguments are presented in Table 2. The encoder plug-in is called “fcc-enc” in the GStreamer framework.

Table 2 *FlexCode* encoder "fcc-enc" interface to GStreamer.

Control parameter	**Value range**	**Notes**	**Default**
Coding	0, 1	Coding mode 0=KLT, 1=MDCT	0
Mode	12 - 64	Local encoder bit rate	24
Request	12 - 64	Far end encoder bit rate	24
Dploss	0 – 0.5	Local design packet loss rate	0.01
Dplreq	0 – 0.5	Far-end design packet loss rate	0.01
Mdcbtcod	0, 1	Selected coding 0=MDC, 1=bit-plane	0
Mdcbtreq	0, 1	Requested coding 0=MDC, 1=bit-plane	0

The *FlexCode* encoder plug-in can be operated as "stand-alone" encoder within the GStreamer framework as follows:

```
> gst-launch filesrc location=input_mono_16kHz.wav ! wavparse ! fcc-enc
coding=0 mode=12 dploss=0.01 ! filesink location=fcc_encoded_binary.bin
```

In this example, the input arguments to the "fcc-enc" encoder set the encoding rate to12 kbit/s using the KLT mode. The encoder adapts the MDC encoding based on the design frame error rate of 1%.

4.2.2 Transmitter

The media handling and transport on a transmitting PC client side is presented in Figure 3. The speech or audio is captured with a standard ALSA audio device after which the content is encoded with the *FlexCode* source and channel encoder. The resulting bit stream is written to the TCP or UDP port and transmitted over an IP network to the receiver. The application only needs to know the IP address and port number of the receiving client.

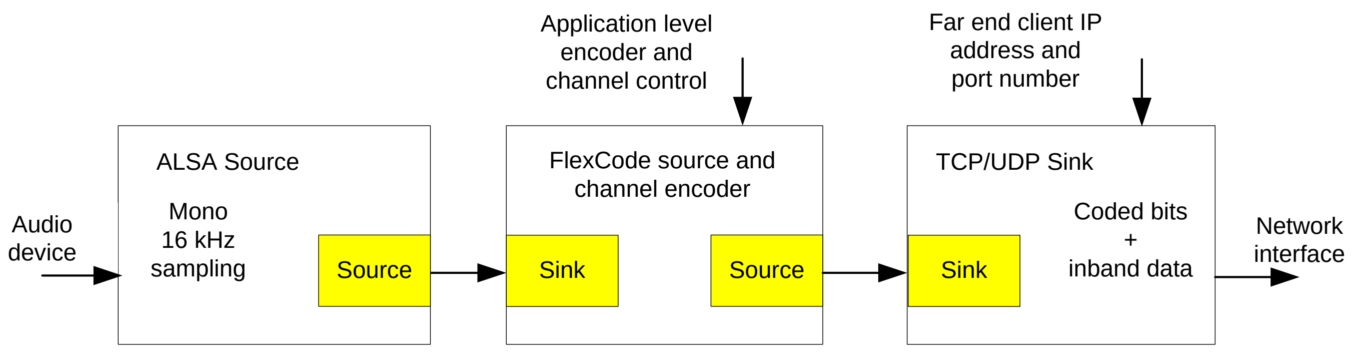
Application level encoder and channel control
Far end client IP address and port number
Audio device
ALSA Source
Mono 16 kHz sampling
Source
FlexCode source and channel encoder
Sink
Source
TCP/UDP Sink
Sink
Coded bits + inband data
Network interface

Figure 3 Encoding and transmission of 16 kHz mono signal with *FlexCode* source and channel encoder within GStreamer framework.

4.2.3 Source and channel decoder and channel model

Figure 4 presents an overview of the *FlexCode* decoder element in the GStreamer framework consisting of source and channel decoding tools, channel model simulation, and in-band data receiver. The *FlexCode* channel model simulation was integrated into the decoder element to balance the overall complexity and to simplify the demonstrator architecture. The input to the element is the coded bit stream from the *FlexCode* source and channel encoder. In addition, the data stream contains in-band signalling. The control parameters in the in-band signalling need to be extracted from the stream and forwarded as output arguments of the plug-in element for the application. The remaining coded bits are then forwarded through the channel model simulation in which bit errors as well as packet errors are inserted based on the given channel conditions. The coded bits are then decoded with the *FlexCode* channel and source decoder. The output of the elements is 16 kHz mono audio signal in 16 bit linear format.

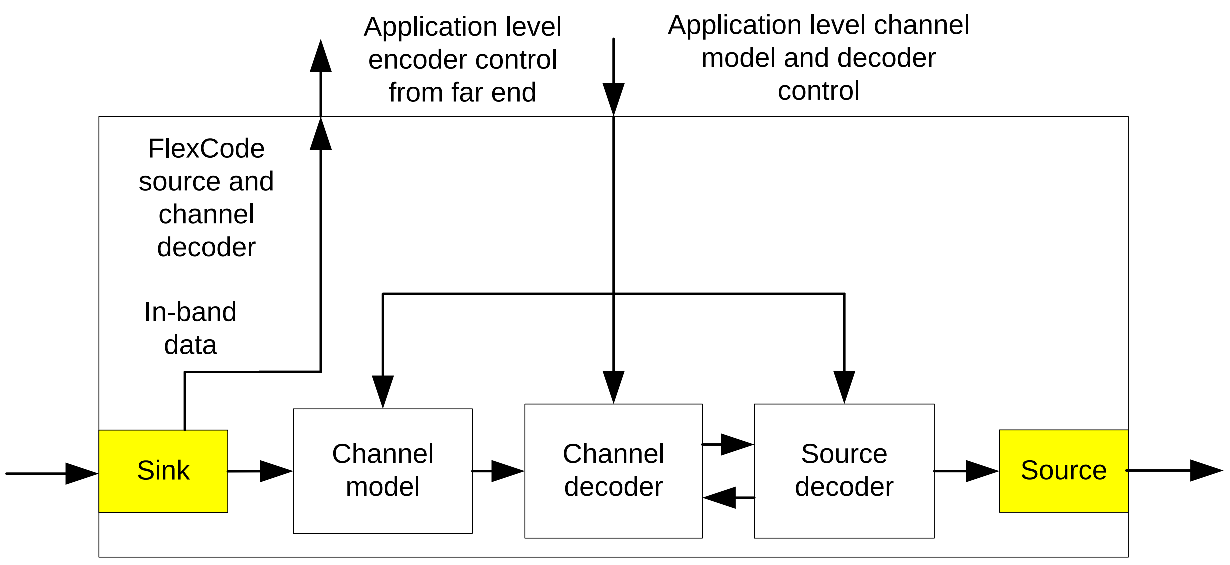
Application level encoder control from far end
Application level channel model and decoder control
FlexCode source and channel decoder
In-band data
Sink
Channel model
Channel decoder
Source decoder
Source

Figure 4 *FlexCode* decoder element with application level control from far-end for the encoder in the local client.

The GStreamer decoder implementation has several input arguments. The main difference to encoder interface is that the decoder also has output arguments retrieved from the in-band data. The interface details are presented in Table 3. The decoder plug-in is called “fcc-dec” in the GStreamer framework.

Table 3 *FlexCode* decoder “fcc-dec” interface to GStreamer.

Control parameter	**Value range**	**Notes**	**Default**
Coding	0, 1	Input: Coding mode 0=KLT, 1=MDCT	0
Request	12 – 64	Output: Local encoder bit rate	24
Aploss	0 – 0.5	Input: Actual packet loss rate	0.01
Rploss	0 – 0.5	Output: Requested design packet loss rate	0.01
Desno	-10 – 40	Input: Design ES/N0	30
Aesno	-10 – 40	Input: Actual ES/N0	30
Mdcbtreq	0, 1	Output: Requested coding 0=MDC, 1=bit-plane	0
Decbitrate	0-50	Input: Removed number of bits in decoder	0

The *FlexCode* encoder plug-in can be operated as a “stand-alone” decoder within the GStreamer framework as follows:

```
> gst-launch filesrc location= fcc_encoded_binary.bin ! fcc-dec
coding=0 desno=10 aesno=30 aploss=0.02 ! wavenc ! filesink
location=fcc_decode_wav.wav
```

In this example, the input arguments to “fcc-dec” decoder select the KLT mode and set the actual frame error rate of the channel to 2%, the channel simulation is set to ES/N0 = 30 dB while the channel decoder is set to handle channel conditions of ES/N0 = 10 dB.

4.2.4 Receiver

Media handling on the receiver side is just the opposite from the transmitter implementation. The GStreamer pipeline consisting of the *FlexCode* decoding element reads the TCP or UDP port and receives the coded bit stream and in-band data from the transmitting client. The *FlexCode* plug-in element extracts the in-band data and provides the control information for the application. For example, the application reads the encoder bit rate requests and forwards them to the encoding element running in the same client. The channel simulation and decoding pipeline is presented in Figure 5. The local decoder control is handled as input arguments to the element. For example, the channel conditions are controlled with these arguments.

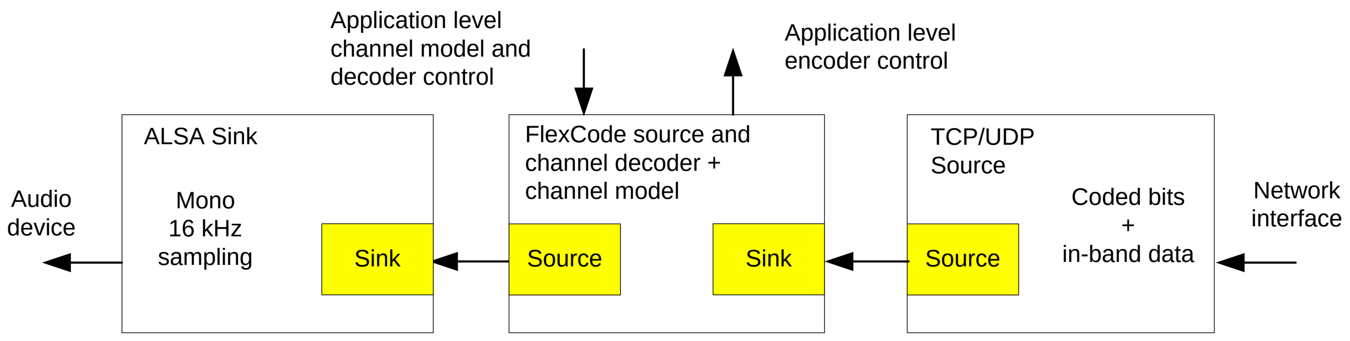
Application level channel model and decoder control
Application level encoder control
ALSA Sink
Mono 16 kHz sampling
Sink
FlexCode source and channel decoder + channel model
Source
Sink
TCP/UDP Source
Coded bits + in-band data
Source
Audio device
Network interface

Figure 5 Channel simulation and decoding of 16 kHz mono signal with *FlexCode* source and channel decoder within GStreamer.

5 Demonstration platform

The generic *FlexCode* demonstration platform consists of two VoIP clients equipped with a network connection, a microphone for audio capture and a headset for audio representation. The IP data transmission between the clients is arranged with either Ethernet cable or WLAN.

Each VoIP client is running both the encoder and the decoder simulation simultaneously. Hence, the GStreamer pipelines presented in Figure 3 and Figure 5 are operated by the demonstration application. The main task of the application is to handle the graphical user interface, take care of the control signals and manage the GStreamer pipeline operation. In addition, the in-band messages between encoding and decoding pipelines are synchronised.

5.1 Graphical user interface (GUI)

The main control parameters of the *FlexCode* demonstrator were implemented on the graphical user interface as sliders. They enable easy modification of the codec bit rate, coding mode, channel coding properties and transmission channel conditions. The message window provides information about the ongoing VoIP session, IP address and port number as well as the selected codec state such as bit rate and design channel conditions.

The far-end client IP address and port number details as well as the start and close of the application are provided as input to the demonstration with drop down menus of the GUI.

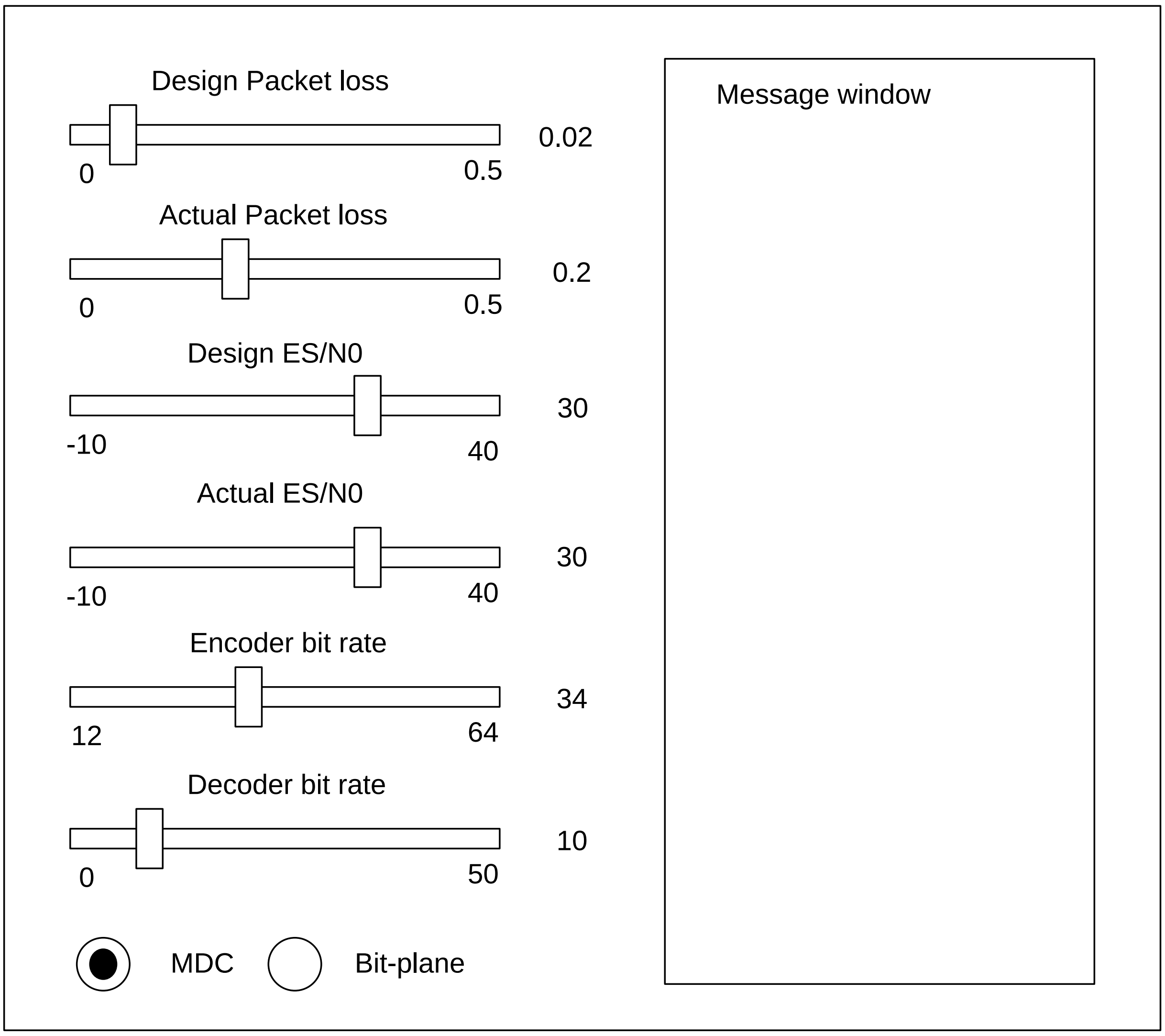

Figure 6 Graphical user interface of the *FlexCode* demonstrator.

5.2 User interface and application

The *FlexCode* demonstration application behind the graphical interface maps the control signals to the GStreamer pipelines. Table 4 lists the control parameters and corresponding *FlexCode* GStreamer plug-in implementation input argument. It should be noted that the selection between KLT and MDCT coding is not available in the GUI. Since switching between different coding modes is not possible during the simulation, the KLT/MDCT mode needs to be selected before starting the application. Therefore, separate applications were built to handle KLT and MDCT coding modes.

Table 4 Mapping of GUI parameters to *FlexCode* encoder and decoder implementation.

GUI parameter	**Encoder “fcc-enc”**	**Decoder “fcc-dec”**	**Control target**
Design packet loss	Rploss	-	Far-end encoder
Actual packet loss	-	Aploss	Local decoder
Design ES/N	-	Desno	Local channel decoder
Actual ES/N0	-	Aesno	Local channel simulation
Encoder bit rate	Request	-	Far-end encoder
Decoder bit rate	-	decbitrate	Local decoder
MDC / Bit-plane	Mdcbtreq	-	Far-end encoder

Table 4 clearly shows that all the GUI parameters intended for the encoder actually control the encoder in the far-end client using in-band data transport, while the decoder and channel control changes the local decoder and channel simulation operation. As discussed earlier, the application ensures that the encoder control requests are retrieved from the received bit stream and forwarded to the encoder.

5.3 Software framework

The *FlexCode* demonstrator is built on a Linux based Maemo platform. The advantage is the availability of open source tools and compatibility between Linux PCs and, e.g., Nokia N800/N810 internet tablets. Hence, the demonstration platform can easily be extended to mobile devices. The same code base can be used on both architectures without any modifications. A streaming type demonstrator can be realised using the mobile device as a receiving client. However, for complexity reasons, the VoIP demonstration is only available for a PC implementation.

The development platform and SDK used for the creation of the demonstration platform is available in http://maemo.org/development. Maemo 4.1.2 Diablo was applied to create the graphical user interface, data transport and audio media handling.

5.4 Hardware architecture

The *FlexCode* demonstration platform consists of two PC clients connected to each other over a fixed network, a WLAN, using ad-hoc networking over WLAN as presented in Figure 7. This architecture can be applied for real-time conversation (VoIP) as well as an audio streaming type of demonstration. In a typical conversational demo setup, the audio is captured with external or PC laptop mounted microphones. The audio is processed and coded within the PCs using the GStreamer media frame implementation. Since the demonstration platform does not have any echo control, the audio presentation needs to be made with a headset to avoid audio signal leakage pack to microphone.

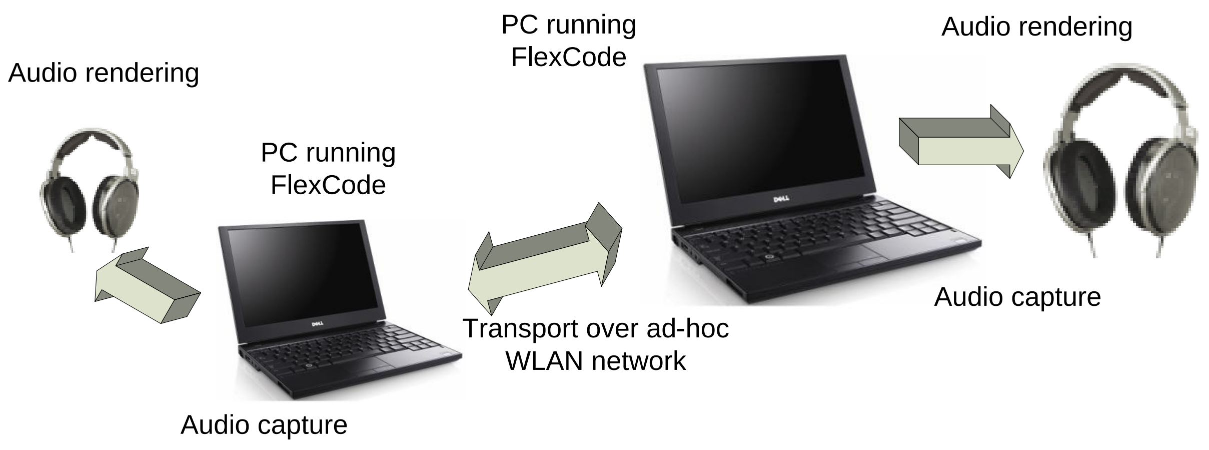

Figure 7 FlexCode hardware demonstration architecture.

6 Conclusions

This report provided a detailed description of the integration of the *FlexCode* source and channel codec into the hardware demonstration. The demonstration platform enables the user to assess the *FlexCod*e codec approach in real-time conversational (VoIP) application under heterogeneous channel conditions and with various audio content. The report explains how the codec flexibility and functionality can be demonstrated in a convenient manner by controlling the main features of the *FlexCode* encoder, decoder and channel model simulation with the graphical user interface.

The technology selection for the demonstration using open source Linux operating system with Maemo platform and SDK was explained. The real-time media handling and data transport protocols with GStreamer framework and integration of *FlexCode* were reviewed in detail.