Features

- **Software Module Dedicated to Voice Processing**
- **Optimized for the AT75 Series Smart Internet Appliance Processor (SIAP™)**
- **Includes Several Run-time Configurable Stand-alone Algorithms**
  - **G.723.1 Dual-rate Vocoder (5.3 Kbps/6.4 Kbps)**
  - **VAD/CNG Silence Compression (Annex A of G.723.1)**
  - **G.711 µ-law and A-law Compression (64 Kbps)**
  - **Arbitrary Tone Generator**
- **ITU-T G.723.1 and G.711 Standard-compliant**
- **Available with a uClinux® Device Driver**

Overview

The AT75C1210 G.723.1 Voice Processing Software Module is designed to run on the OakDSPCore® subsystem of the AT75 series Smart Internet Appliance Processor. It implements commonly-used voice processing algorithms:

- a low bit-rate G.723.1 vocoder for multimedia communication
- a silence compression algorithm to efficiently handle periods of silence during communication
- a high bit-rate voice compression algorithm
- an arbitrary tone generator that can be used to generate any frequency during a programmable duration

All these algorithms have a number of parameters which can be programmed at run time. These parameters modify the behavior of the DSP algorithms in such a manner that they comply with the applicable standards under most situations. They also allow the AT75C to cope with many non-standard situations often encountered on private telephone networks.

The AT75C1210 takes advantage of the AT75 mailbox to exchange data with the on-chip ARM7TDMI® core. The organization of the data communication channel makes it easy to integrate the AT75C1210 interface into most operating systems.

For developers using uClinux, a specific device driver is supplied. It allows the extension of uClinux capabilities to the complete functionality of the AT75C1210 module in a seamless manner.

This document is made up of three sections:

1. a functional description of the supported algorithms
2. a description of the low level software interface
3. a description of the uClinux device driver

Mixing low-level and driver-level programming should be avoided.

Smart Internet Appliance Processor (SIAP™)

AT75C1210 G.723.1 – Voice Processing Software Module

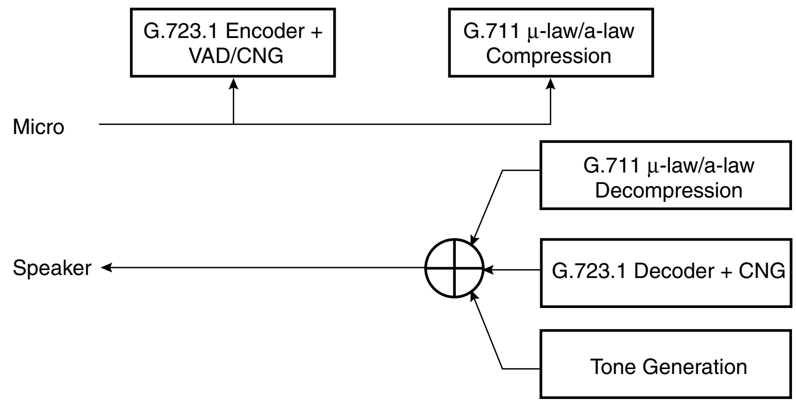
Rev. 1777A–11/01

Functional Description

A functional block diagram of the AT75C1210 G.723.1 module is given in Figure 1.

The different algorithms are independent. They can be enabled, disabled or programmed individually.

Figure 1. Block Diagram



G.723.1 Dual Rate Vocoder

This algorithm can be used for compressing the speech or other audio signal components of a multimedia service at a very low bit rate. This coder has two bit rates associated with it: 5.3 and 6.4 Kbps. The higher bit rate has better quality; it is based on Multi Pulse Maximum Likelihood Quantization (MP-MLQ) technique. The lower bit rate gives good quality and provides system designers with additional flexibility; this rate is based on an Algebraic Code Linear Prediction (ACELP) technique.

This coder operates on 30 ms speech frames of 16-bit linear PCM samples (sampling frequency is 8 kHz). An algorithmic delay of 7.5 ms is to be taken into account before getting an encoded voice data frame. That leads to a total delay of 37.5 ms. Resulting encoded frames are 20 bytes long for 5.3 Kbps rate and 24 bytes long for 6.4 Kbps rate. The encoding rate can be chosen by means of a configuration command sent to the DSP (see “Request Notification Messages” on page 7).

VAD/CNG

Voice Activity Detection (VAD) and Comfort Noise Generator (CNG) algorithms are designed to work hand-in-hand with G.723.1 vocoder. Silence compression techniques are used to reduce the transmitted bit rate during silent intervals of speech. The VAD side detects those silent intervals. CNG is used to produce a noise that matches the actual background noise. CNG uses information provided by VAD to encode silent intervals into Silence Insertion Descriptor (SID) frames that are 4 bytes long. It also re-synthesizes 16-bit linear PCM samples of background noise with a SID frame input. The VAD/CNG feature can be enabled or not by means of a configuration command sent to the DSP (see “Request Notification Messages” on page 7).

G.711 µ-law and A-law Voice Compression

µ-law and a-law are logarithmic compression techniques applied to speech signals. They are done by simple operations that give no delay and excellent quality of speech. However, the bit rate is very high (each 16-bit linear PCM speech sample gives an 8-bit compressed sample leading to 64 Kbps) making this feature useful only for broadband data networks. The compression/decompression algorithm can be chosen by means of a configuration command send to the DSP (see “Request Notification Messages” on page 7).

Tone Generator

The tone generation task generates a pure sine wave with programmable frequency, amplitude and duration.

Low-level Interface

This section describes how the AT75C1210 software is uploaded into the DSP subsystem program memory. It also describes how the application software running on the ARM® and the AT75C1210 running on the DSP Subsystem exchange information through the mailboxes.

This section assumes an in-depth knowledge of the ARM/DSP Subsystem interface mailbox system.

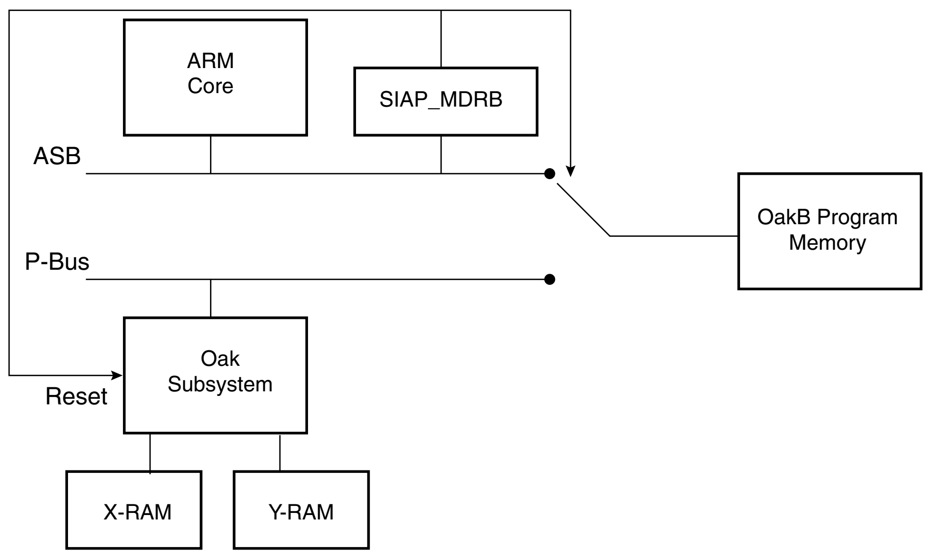
Voice Module Upload

While the DSP subsystem is held in reset, its program memory is made visible in the ARM memory space. This allows the ARM application to write a binary image of the DSP software very easily.

When the DSP subsystem is taken out of reset, its program memory is switched from the ARM memory space back to the DSP program space just before the first instruction is fetched.

This process is illustrated in Figure 2.

Figure 2. Voice Module Upload



Upload Process

A typical DSP program uses a number of initialized variables. Typically, the initial values are stored in the program space, and copied into their RAM location by the DSP start-up routine. This leads to the following statements:

- Just after the boot routine has initialized the variables, the DSP subsystem exhibits high redundancy since the same values exist in both program and data memories.
- The initial values stored in the program memory waste space and are not used during operation.
- To improve the program memory usage, the software is loaded in two consecutive steps.
- A small data initialization program is first loaded and executed. This program just initializes the X- and Y-RAM to the values expected by the audio decoder software. When the initialization is done, the program sends a DATA_INIT_DONE status message to the ARM application through the status mailbox.
- Then, the DSP subsystem is put in reset and the program itself is loaded. This code has no data init start-up routine. It assumes the RAMs are already initialized, which saves program space. When the software is ready to work, it sends a SW_INIT_DONE status message through the status mailbox.

The mailbox operation and status messages are described in the section “Mailbox Usage” on page 5.

Binary Image Format

When the system is idle, the AT75C1210 module is stored in the ARM memory space, possibly in non volatile memory. The module contains the data initialization code, the application code, and additional formatting data. The various fields of the AT75C1210 binary image are described in Table 1.

Table 1. Binary Image Fields

Field Name	Offset from Start of Field (Bytes)	Length (Bytes)	Description
INIT_OFFSET	0	4	Defines the position of the data initialization code from the beginning of the module image.
INIT_LENGTH	4	4	Defines the length of the data initialization code (16-bit words). Valid between 0 and 24576.
SW_OFFSET	8	4	Defines the position of the audio decoder program from the beginning of the module image.
SW_LENGHTH	12	4	Defines the length of the audio decoder code (16-bit words). Valid between 0 and 24576.
INIT_CODE	16	2*INIT_LENGTH	Binary code of the data initialization program.
SW_CODE	16 + 2*INIT_LENGTH	2*SW_LENGTH	Binary code of the application program.

DPMB Configuration

The DPMB is programmed in configuration 2 (as defined in the AT75 Series Datasheet) and gives the configuration shown in Table 2. All the mailboxes allow read/write access from both sides. Arbitration is done using the semaphores.

Table 2. DPMB Configuration

Mailbox #	Offset from Base(1)	Length	Direction	Semaphore Address(1)	Usage
0	0x000	0x80	ARM -> Oak	0x200	Unused
1	0x080	0x80	ARM <- Oak	0x204	Unused
2	0x100	0x40	ARM -> Oak	0x208	DSP memory access
3	0x140	0x40	ARM -> Oak	0x20C	Unused
4	0x180	0x20	ARM -> Oak	0x210	Unused
5	0x1A0	0x20	ARM <- Oak	0x214	Unused
6	0x1C0	0x20	ARM -> Oak	0x218	Request notification
7	0x1E0	0x20	ARM <- Oak	0x21C	Status notification

Note: 1. Base address is 0xfa000000 for OakA, 0xfb000000 for OakB.

Mailbox Access

ARM-to-Oak Mailboxes

Before accessing the ARM-to-Oak mailboxes, the ARM must check that the corresponding semaphore is cleared to 0. Then it can read or write the mailbox data. When the data access is done, it must set the semaphore to 1 to notify the Oak that new data has arrived.

Oak-to-ARM Mailboxes

The ARM is notified that new data is available in a mailbox when the corresponding semaphore is raised to 1, possibly triggering an interrupt. Then the ARM can access the mailbox. When the access is finished, the ARM must clear the semaphore to release the mailbox.

Mailbox Usage

This section describes the specific purpose of each mailbox. The exchanged information is formatted in structured messages. The message format and semantics are described in sections “Request Notification Messages” on page 7 and “Status Notification Messages” on page 11.

Mailbox 0: TX Encoded Voice Data

Used by the ARM to provide to the OAK encoded speech frames (either G.711 data or G.723.1 data).

Mailbox 1: RX Encoded Voice Data

Used by the ARM to get from the OAK encoded speech frames (either G.711 data or G.723.1 data).

Mailbox 2: Oak Memory Access

The ARM has the ability to send requests to read or write any location of the DSP memories, either in program or data space. This is useful for two purposes:

- DSP software debug
- Programming of the DSP peripherals under the ARM application control

Mailbox 6: Request Notification

This mailbox is used by the ARM to pass requests to the DSP. These requests trigger specific tasks in the DSP software. For example, request notification messages are used to start or to stop the telephony algorithms.

Mailbox 7: Status Notification

This mailbox is used by the DSP software to send status information. For example, a status notification message is sent by the DSP software at the end of the data initialization to notify the ARM application that the data has been initialized.

TX/RX Encoded Voice Data

The first two mailboxes deal with speech compressed frames. Each byte sent through the mailbox is put in a 16-bit word where the low byte is the original byte value and in the high byte are flags.

Assuming the data to be transmitted is in “char buf[0..N-1]”, it is formatted in the mailbox as shown in Table 3 (otherwise the frame is ignored).

Table 3. Speech Frame Format(1)

Word 0	...	Word i (i = 1... N - 2)	...	Word N - 1
FRAME_START|buf[0]	...	0x0000|buf[i]	...	FRAME_END|buf[N - 1]

Note: 1. With FRAME_START = 0x8000 and FRAME_END = 0x4000

Delivered frames are of variable length:

- The length is encoded within the first two bits of buf[0] for G.723.1:
  - buf[0] & 0x3 = 0 -> 24 bytes at a rate of 6.3 Kbits per second
  - buf[0] & 0x3 = 1 -> 20 bytes for a rate of 5.3 Kbits per second
  - buf[0] & 0x3 = 2 -> 4 bytes for silence compression frames

– buf[0] & 0x3 = 3 -> 1 byte: it follows a 4-byte frame while the silence scheme is unchanged

- If the system is in G.711, mode frames are 64 bits long, independent of the contents of buf[0].

Oak Memory Access

The ARM has the ability to send requests to read or write any location of the Oak memories, either in program or data space. To achieve this, the mailbox 2 is divided into four fields:

- Command field (mailbox base + 0): This is a request ID that tells what kind of operation is to be performed. Valid codes are:
  – 0x0001: Program memory read
  – 0x0002: Program memory write
  – 0x0003: Data memory read
  – 0x0004: Data memory write
- Address field (base + 1 16-bit word): Should be written with the address location to be accessed. This is the value of the address as it is seen by the Oak.
- Length field (base + 2 16-bit words): Should be written with the number of consecutive locations to access.
- Data field (base + 3 16-bit words and following): For write access, should be filled with the values to write. For read access, contains the read values requested by the previous command.

Example of use: Write 0x1234 into data location 0xabcd of the 0akB:

1. Wait for *(0xfb000208) == 0, i.e., the semaphore is cleared
2. *(0xfb000100) = 0x0004 // data write command
3. *(0xfb000102) = 0xabcd // this is the address
4. *(0xfb000104) = 0x0001 // only one word to write
5. *(0cfb000106) = 0x1234 // this is the value
6. *(0xfb000208) = 1 // notify the OakB

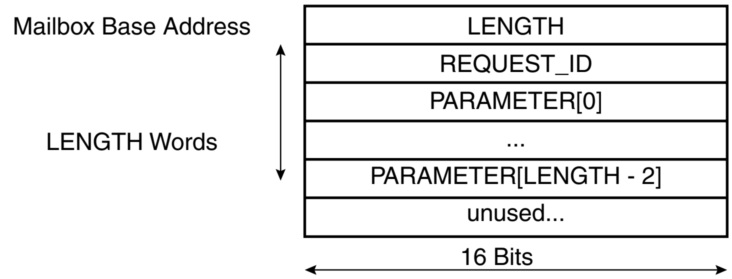
Example of use: Read data locations 0xabcd and 0xabce from OakB:

1. Wait for *(0xfb000208) == 0, i.e. the semaphore is cleared
2. *(0xfb000100) = 0x0004 // data write command
3. *(0xfb000100) = 0x0003 // data read command
4. *(0xfb000102) = 0xabcd // this is the first address to read
5. *(0xfb000104) = 0x0002 // two words to read
6. *(0xfb000208) = 1 // notify the OakB
7. Wait for the semaphore to go back to 0.
8. Read 0xfb000106 and 0xfb000108 to get the requested values.

Request Notification Messages

Request messages are used by the ARM to trigger specific tasks running on the DSP. These messages are always formatted in the same way. Figure 3 describes this format.

Figure 3. Request Notification Message Format



A message always begins with a LENGTH field. This field contains the number of words of the message, excluding the LENGTH field itself.

The REQUEST_ID field is uniquely defined to designate the type of request. Each request can be followed by a variable but well-defined number of PARAMETER fields. These fields contain additional data needed to handle the request.

The description of the supported request messages is listed inTable 4. It is forbidden for the ARM application to issue unsupported messages. However, should the ARM application issue an unsupported or malformed request, the Oak software must recover gracefully.

G.723.1 Configuration Request

This message is sent to the Oak before enabling any G.723 operation.

Table 4. G.723.1 Configuration Request

Word 0	0x0006	Message Length = 0x0006
Word 1	0x0400	Request ID = 0x0400
Word 2	WORKRATE	Work rate for encoding, valid values: 0: 6.3 Kbits/s rate 1: 5.3 Kbits/s rate
Word 3	WORKRATED	Work rate for decoding, valid values: 0: 6.3 Kbits/s rate 1: 5.3 Kbits/s rate default 1 Note: This parameter is automatically set to the G.723 algorithm. In all cases this parameter needs to be initialized.
Word 4	USEVX	0: disable VAD 1: enable VAD
Word 5	MICR_GAIN = 0x1000 * 10E(dB/20)	Gain for the microphone input Valid: 0x0040 (- 36 dB) to 0x8000 (+18 dB)
Word 6	SPKR_GAIN = 0x1000 * 10E(dB/20)	Gain for the speaker output Valid: 0x0040 (-36 dB) to 0x8000 (+18 dB)

G.723.1 Decoding Start Request

Table 5. G.723.1 Decoding Start Request

Word 0	0x0001	Message length = 0x0001
Word 1	0x0401	Request ID = 0x0401

The G.723.1 decode task starts as soon as the DSP unit receives this request.

G.723.1 Decoding Stop Request

Table 6. G.723.1 Decoding Stop Request

Word 0	0x0001	Message length = 0x0001
Word 1	0x0402	Request ID = 0x0402

The G.723.1 decode task is stopped as soon as this request is received by the DSP unit.

G.723.1 Encoding Start Request

Table 7. G.723.1 Encoding Start Request

Word 0	0x0001	Message length = 0x0001
Word 1	0x0403	Request ID = 0x0403

The G.723.1 encode task starts as soon as the DSP unit receives this request.

G.723.1 Encoding Stop Request

Table 8. G.723.1 Encoding Stop Request

Word 0	0x0001	Message length = 0x0001
Word 1	0x0404	Request ID = 0x0404

The G.723.1 encode task is stopped as soon as this request is received by the DSP unit.

G.711 Configuration Request

Table 9. G.711 Configuration Request

Word 0	0x0005	Number of words of the message
Word 1	0x0401	Request ID
Word 2	LAW	Selected Law for compression. Valid values: 0: µ-law 1: a-law default 0

Table 9. G.711 Configuration Request (Continued)

Word 3	LAWD	Selected Law for decompression. Valid values: 0: μ-law 1: a-law default 0
Word 4	MICR_GAIN = 0x1000 * 10E(dB/20)	Gain for microphone input Valid: 0x0040 (-36 dB) to 0x8000 (+18 dB)
Word 5	SPKR_GAIN = 0x1000 * 10E(dB/20)	Gain for the speaker output Valid: 0x0040 (-36 dB) to 0x8000 (+18 dB)

This message is sent to the Oak before enabling any G.711 operation

G.711 Decompression Start Request

Table 10. G.711 Decompression Start Request

Word 0	0x0001	Number of words of the message
Word 1	0x0411	Request ID

The G.711 decompression task starts as soon as the DSP unit receives this request.

G.711 Decompression Stop Request

Table 11. G.711 Decompression Stop Request

Word 0	0x0001	Number of words of the message
Word 1	0x0412	Request ID

The G.711 decompression task is stopped as soon as the DSP unit receives this request.

G.711 Compression Start Request

Table 12. G.711 Compression Start Request

Word 0	0x0001	Number of words of the message
Word 1	0x0413	Request ID

The G.711 compression task starts as soon as the DSP unit receives this request.

G.711 Compression Stop Request

Table 13. G.711 Compression Stop Request

Word 0	0x0001	Number of words of the message
Word 1	0x0414	Request ID

The G.711 compression task is stopped as soon as the DSP unit receives this request.

Tone Generation Configuration Request

Table 14. Tone Generation Configuration Request

Word 0	0x0007	Message Length = 0x0007
Word 1	0x0800	Request ID = 0x0800
Word 2	32768 * cos (pi * TONE_FREQ/4000)	Words 2 and 3 define the frequency of the generated tone
Word 3	32768 * cos (pi * TONE_FREQ/4000)	
Word 4	TONE_LEVEL = 32768 * 10E(dB/20)	Level of the generated tone
Word 5	TONE_DURATION	Duration of the generated tone in milliseconds 0x0000 means unlimited duration
Word 6	SILENCE_DURATION	Duration of the silence following the tone in milliseconds 0x0000 means unlimited duration
Word 7	TONE_START	Bit 0: 0 causes the generator to wait for a tone generation start request (request ID 0x0801) before the tone is generated 1: the generation starts immediately Bit 1: 0: the tone is added to all other signals emitted on the speaker 1: all other signals are blocked while the tone is generated

Example: 0x0007 0x0801 0x5A82 0x5A83 0x4000 0x0080 0x0080 0x0003

This message configures the generator to emit a 1024 Hz tone 6 dB below the reference level. The tone is emitted as soon as the DSP unit receives the request. After 128 ms of signal and 128 ms of silence, a tone generation done status message is emitted.

Tone Generation Start Request

Table 15. Tone Generation Start Request

Word 0	0x0001	Message length = 0x0001
Word 1	0x0801	Request ID = 0x0801

The tone starts as soon as the DSP unit receives this request.

A tone generation configuration request (request ID 0x0800) should be issued before the tone generation start request is sent. If not, the behavior of the tone generator is unpredictable.

Tone Generation Stop Request

Table 16. Tone Generation Stop Request

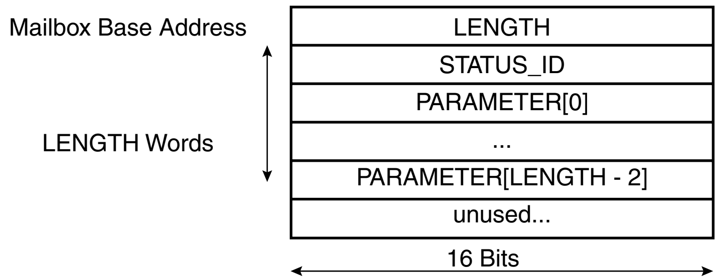
Word 0	0x0001	Message length = 0x0001
Word 1	0x0802	Request ID = 0802

The tone stops as soon as the DSP unit receives this request. This request can be used to stop an unlimited tone generation, or to halt the generator before the predefined duration has elapsed (early termination).

Status Notification Messages

Status messages are used by the Oak to inform the ARM application that a specific event has occurred, or to respond to an earlier request. These messages are always formatted in the same way. Figure 4 describes this format.

Figure 4. Status Notification Message Format



A status message always begins with a LENGTH field. This field contains the number of words of the message, excluding the LENGTH field itself.

The STATUS_ID field is uniquely defined to designate the type of status. Each status can be followed by a variable but well-defined number of PARAMETER fields. These fields contain additional status information.

The description of the supported status messages is listed below. It is forbidden for the Oak program to issue unsupported status messages. However, should the Oak program issue an unsupported or malformed status message, the ARM application must recover gracefully.

Data Initialization Status

This status message is issued when the data initialization program has completed the data initialization process. The Oak can be safely reset and reloaded with the voice module precisely named.

Table 17. Data Initialization Status

Word 0	0x0006	Message length = 0x0006
Word 1	DATA_INIT_DONE_ID	Status ID = 0x8001
Word 2	VERSION_MONTH	Version information: Contains the date of the generation of the binary file of the DSP.
Word 3	VERSION_DAY	
Word 4	VERSION_YEAR	
Word 5	VERSION_HOUR	
Word 6	VERSION_MIN	

Voice Module Initialization Status

This status message is issued when the audio decoder has finished initializing itself and is ready to accept request messages. The ARM should not issue any request messages before this status message has been received.

Table 18. Voice Module Initialization Status

Word 0	LENGTH	Message length = 0x0001
Word 1	SW_INIT_DONE_ID	Status ID = 0x8002

Bad Format Status

The Oak issues this message when it has received a request message in which the LENGTH field is not compatible with the request type. The OakB ignores the corresponding malformed request.

Table 19. Bad Format Status

Word 0	LENGTH	Message length = 0x0002
Word 1	BAD_FORMAT_ID	Status ID = 0x80FF
Word 2	BAD_FORMAT_VALUE	Contains the request ID of the malformed request message.

Unknown Request Status

The Oak issues this message when it has received a request message with an unsupported request ID field.

Table 20. Unknown Request Status

Word 0	LENGTH	Message length = 0x0002
Word 1	UNKNOWN_REQ_ID	Status ID = 0x80FE
Word 2	UNKNOWN_REQ_VALUE	Contains the request ID of the malformed request message.

Bad Parameter Status

The Oak issues this message when it has received a request message with a parameter having an invalid value.

Table 21. Bad Parameter Status

Word 0	LENGTH	Message length = 0x0002
Word 1	UNKNOWN_REQ_ID	Status ID = 0x80FD
Word 2	UNKNOWN_REQ_VALUE	Contains the request ID of the malformed request message.

Bad Speech Frame Status

This status is issued when a speech frame message does not have the correct header or footer

Table 22. Bad Speech Frame Status

Word 0	LENGTH	Message length = 0x0001
Word 1	BAD_FRAME_ID	Request ID = 0x84FF

Underrun Status

This status is issued when a speech frame takes too long to arrive, thus causing a discontinuity in the speech stream.

Table 23. Underrun Status

Word 0	LENGTH	Message length = 0x0001
Word 1	UNDERRUN_ID	Status ID = 0x84FD

To avoid flooding the ARM with underrun status bursts, this kind of message should be issued at most once per compressed frame period, as long as the underrun state is encountered.

G.723.1 Decoding Stopped Status Message

This status is issued if the decode task was stopped by a G.723.1 decode stop request (request ID 0x0402).

Table 24. G.723.1 Decoding Stopped Status Message

Word 0	0x0001	Message length = 0x0001
Word 1	0x8402	Status ID = 0x8402

G.723.1 Encoding Stopped Status Message

This status is issued if the decode task was stopped by a G.723.1 encode stop request (request ID 0x0404).

Table 25. G.723.1 Encoding Stopped Status Message

Word 0	0x0001	Message length = 0x0001
Word 1	0x8404	Status ID = 0x8404

G.711 Decompression Stopped Status Message

This status is issued if the decompression task was stopped by a G.711 decompression stop request (request ID 0x0412).

Table 26. G.711 Decompression Stopped Status Message

Word 0	0x0001	Message length = 0x0001
Word 1	0x8412	Status ID = 0x8412

G.711 Compression Stopped Status Message

This status is issued if the compression task was stopped by a G.711 compression stop request (request ID 0x0414).

Table 27. G.711 Compression Stopped Status Message

Word 0	0x0001	Message length = 0x0001
Word 1	0x8414	Status ID = 0x8414

Tone Generation Status

This message is issued when the tone duration has elapsed. It is not issued if the tone was stopped by a tone generation stop request (request ID 0x0802).

Table 28. Tone Generation Status Message

Word 0	0x0001	Message length = 0x0001
Word 1	0x8802	Status ID = 0x8802

AT75C1210 Device Driver

The AT75C1210 software module is supplied with device driver for uClinux. This device driver enables the application developer to integrate all the AT75C1210 functionality into the uClinux kernel. All the features of the AT75C1210 modules can be accessed through the standard uClinux API. This section documents this API.

Under uClinux, the device drivers are accessed through filesystem entries. The AT75C1210 device driver is a character type driver. The associated virtual file can be opened, read from, written to and closed like any regular file. The major role of the device driver is to redefine the file access methods, so that the application can interact with the underlying device as if it were a file through the standard file manipulation functions. It provides the application with an abstraction layer which hides the low level interface on top of which it sits.

The AT75C1210 device driver is operated through the /dev/g723 filesystem. It is used for G.723.1 operations.

G.723.1 Driver Operations

The G.723.1 driver redefines the following file manipulation functions:

- int open(const char *path, int flags, mode_t mode);
- int read(int fd, void *buf, int count);
- int write(int fd, void *buf, int count);
- int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
- int close(int fd);

Additionally, the ioctl function allows control of additional features of the AT75C1210 that are not accessible with the read or write methods. Those special commands are described below. The prototype of the ioctl function is:

- int ioctl(int fd, int request, char *argp);

Open Method

Synopsis

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int flags);
```

Description

The /dev/g723 virtual file must be opened prior to any operation on the G.723 device driver. This is done with the open method, just like for any regular file. The main operation performed by the open method of the device driver is to load and initialize the corresponding DSP software in the DSP subsystem.

When this initialization is successful, the open system call converts the file path name ("/dev/g723" in this case) into a file descriptor. This file descriptor is a non-negative integer that is used in subsequent I/O operations such as with read, write, etc.

flags is one of O_RDONLY, O_WRONLY or O_RDWR which request opening the file read-only, write-only or read/write, respectively.

flags may also be bitwise-or'd with O_NONBLOCK. In this case, neither the open nor any subsequent operation on the file descriptor which is returned causes the calling process to wait.

Return Values

Open return the new file descriptor, or -1 if an error occurred. In the latter case, the global variable errno is set appropriately to reflect the cause of error. Possible values of errno are:

- ENODEV: this indicates that the underlying hardware does not exist or is not supported. One reason can be a corruption of the binary DSP software which could not be loaded into the DSP subsystem.
- EBUSY: the underlying hardware is busy. Most probably there is another process using the same resource.
- ENOMEM: a memory allocation requested by the driver failed. This happens when the system memory is full.

Example

```
int fd = open("/dev/g723", O_RDWR | O_NONBLOCK);
```

This opens the G.723 device driver in read/write mode. It selects non blocking I/O for read and write operations. The file descriptor is returned in fd. If fd is positive, the G.723 device is readily available for read and write operations.

Close Method

Synopsis

```
#include <unistd.h>
int close(int fd);
```

Description

When the G.723 device is not needed any longer by the application, it can be closed to release system resources. This is done through the close method. The parameter is the file descriptor of the file to be closed.

Return Values

Close returns 0 on success, or -1 if an error occurred. In the latter case the global variable errno is set appropriately to reflect the cause of error. The only possible value for errno is EBADF which means that fd is not a valid file descriptor.

Example

```
close(fd);
```

This closes the G.723 device previously opened.

Read Method

Synopsis

```
#include <unistd.h>
int read(int fd, void *buf, int count);
```

Description

As for any file descriptor, the read method attempts to read count bytes from fd into the buffer starting at buf. When fd is a file descriptor attached to /dev/g723, the bytes read correspond to the frame recognized by the G.723 decoding device.

Both blocking and non-blocking reads are supported. In blocking mode, read returns only when there is a G.723 frame available to read. Although the process is blocked, it is safely put on a system wait queue and does not consume CPU time.

In non-blocking mode, the read function returns immediately even if no data is available. In this case the return value is -1 and errno is set to EAGAIN.

Return Values

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested. This may happen for example because fewer bytes are actually available at the time, or because read was interrupted by a signal.

On error, -1 is returned and errno is set appropriately. Possible values for errno are as follows:

- EAGAIN: non-blocking I/O has been selected using O_NONBLOCK and no data was immediately available.
- EBADF: fd is not a valid descriptor.
- EINVAL: the /dev/g723 file was not open for reading.
- EFAULT: buf is outside the accessible address space.

Example

```
ret = read(fd,buf,256);
```

This reads at most 256 bytes from file descriptor fd (assumed here to be related to /dev/g723), and stores them into the memory location pointed to by buf.

Write Method

Synopsis

```
#include <unistd.h>
int write(int fd, void *buf, int count);
```

Description

As for any file descriptor, the write method attempts to write count bytes from the buffer starting at buf to the file descriptor fd. When fd is a file descriptor attached to /dev/g723, the bytes written correspond to the G.723 frame which is to be emitted by the G.723 device.

Both blocking and non-blocking writes are supported. In blocking mode, write returns only when the G.723 device is ready to accept data. Although the process is blocked, it is safely put on a system wait queue and does not consume CPU time.

In non-blocking mode, the write function returns immediately even if no data is available. In this case the return value is -1 and errno is set to EAGAIN. In most cases the application retries to write until the entire data set is transferred.

Return Values

On success, the number of bytes written is returned. This corresponds to the number of G.723 bytes actually emitted. It is not an error if this number is smaller than the number of bytes requested. This may happen for example because fewer bytes are actually acceptable at the time due to lack of memory, or because write was interrupted by a signal.

On error, -1 is returned and errno is set appropriately. Possible values for errno are as follows:

- EAGAIN: non-blocking I/O has been selected using O_NONBLOCK and no data was immediately available.
- EBADF: fd is not a valid descriptor.
- EINVAL: the /dev/g723 file was not open for reading.
- EFAULT: buf is outside the accessible address space.

Example

```
ret = write(fd,buf,256);
```

This writes at most 256 bytes to file descriptor fd (assumed here to be related to /dev/g723), from the memory location pointed to by buf.

Ioctl Method

Synopsis

```
#include <sys/ioctl.h>
int ioctl(int fd, int request, char *argp);
```

Description

The ioctl function manipulates the underlying device parameters of the G.723 device.

fd is the file descriptor upon which ioctl acts. It is related to the /dev/g723 virtual file.

request defines which predefined command to send to the G.723 device. Some commands may require additional arguments which are stored or received in the buffer pointed to by argp. The ioctl requests supported by the G.723 device driver are described below:

- G723_START_PLAYBACK: This command is used to start the G.723 playback. There is no additional argument.
- G723_STOP_PLAYBACK: This command is used to stop the G.723 playback. There is no additional argument.
- G723_START_RECORD: This command is used to start the G.723 record. There is no additional argument.
- G723_STOP_RECORD: This command is used to stop the G.723 record. There is no additional argument.
- G723_CONFIG: This command is used to configure the characteristics of the G.723 vocoder algorithm. An additional parameter is used as defined below:

```
struct config_args {
unsigned short enc_rate;
unsigned short dec_rate;
unsigned short vad_cng;
unsigned short mic_gain;
unsigned short spk_gain;
};
```

The fields and the values to be written are those defined in the section on “Low-level Interface” on page 3.

Example

```
struct config_args {
unsigned short enc_rate;
unsigned short dec_rate;
unsigned short vad_cng;
unsigned short mic_gain;
unsigned short spk_gain;
} *g723_conf;
g723_conf->enc_rate=0;//6.3 rate for coder
g723_conf->dec_rate=0;//6.3 rate for decoder
g723_conf->vad_cng=0;//no VAD/CNG
g723_conf->mic_gain=4096;//micro gain 0dB under reference
g723_conf->spk_gain=4096;//speaker gain 0dB under reference
ioctl(g723, G723_CONFIG, g723_conf);
```

This configures the G.723 algorithm.

Installation

For versions of the siap_uClinux previous to 2.0, the installation of the AT75C1210 software is as follows:

Change directory to `siap-uClinux-1.x.y/` and launch `patch_AT75C1210`. It carries out the following actions:

- Add `g723.bin` DSP binary in the `prods/dk020/romdisk/romdisk/lib/` directory.
- Add `voice/` demo sources subdirectory in `apps/` directory
- Add `g723/` driver subdirectory in `linux/arch/armnommu/driver/` directory
- Modify various configuration files

After it ends, change directory to `linux/` and type:

```
> make xconfig
```

This updates the configuration according to the file modification. Verify that the “G.723.1 support” item is correctly set to “y”. Afterwards clean and rebuild your uClinux distribution.

For versions 2.0 and higher, the driver is already installed.

Application Example

Synopsis

```
#include <asm/messages.h>
```

The demo application delivered with AT75C1210 driver illustrates its capabilities.

Start a G.723 Recording

On the board type:

```
> voice -rec <your_file>
```

This opens the G.723 device and records about 10 seconds of voice.

Start a G.723 Playback

On the board type:

```
> voice -play <your_file>
```

This opens the G.723 device and plays back the specified file.

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Product Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

Atmel Heilbronn
Theresienstrasse 2
POB 3535
D-74025 Heilbronn, Germany
TEL (49) 71 31 67 25 94
FAX (49) 71 31 67 24 23

Atmel Nantes
La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 0 2 40 18 18 18
FAX (33) 0 2 40 18 19 60

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Atmel Smart Card ICs
Scottish Enterprise Technology Park
East Kilbride, Scotland G75 0QR
TEL (44) 1355-357-000
FAX (44) 1355-242-743

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

Printed on recycled paper.

1777A–11/01/0M