Development of a USB Telephony Interface Device for Speech Recognition Applications

J.J. Müller and T.R Niesler
Department of Electric and Electronic Engineering, University of Stellenbosch, Stellenbosch.
Tel: (021) 8084315, Fax: (021) 8083951, E-mail: {jjmuller, trn}@dsp.sun.ac.za

Abstract— Automatic Speech Recognition (ASR) systems are rapidly replacing Interactive Voice Response (IVR) systems, as an attractive means for companies to deliver value added services with which to improve customer satisfaction. Such ASR systems require a Telephony Interface to connect the speech recognition application to the Public Switched Telephone Network (PSTN). Commercially available telephony interfaces are usually complex and expensive devices whose drivers and API's are often available only for the Microsoft Windows operating systems. This poses a problem, as many of the tools used for speech recognition research and development operate on LINUX-based systems. This paper describes the design and development of a USB-based telephony interface offering cross-platform portability.

Index Terms— Application Programming Interface (API), platform-independence, Public Switched Telephone Network (PSTN) Interface, Universal Serial Bus, Speech Recognition.

I. INTRODUCTION

Speech is the most natural form of interaction for people, and it would thus make sense to develop technology that would make it possible for human beings to interact with computers by using speech. If speech and human language can be used as a computer interface, computer services can be accessed by anyone. This is particularly useful to the illiterate segments of our population. Speech technology could be used in an automated, voice-operated computer system that could provide educational, commercial and public services that would otherwise be inaccessible to them.

A speech-based computer interface can be adapted to cater for indigenous languages, which will facilitate better service delivery as more people will be able to access or supply information in any of the official languages. This will also stimulate development and promote the use of the indigenous languages according to our government's policy of multilingualism.

Telephones are a very important access channel to such automated speech-based systems, since the greatest part of the South African population has access to a telephone. The Digital Signal Processing group and the Department of African Languages at the University of Stellenbosch are collaborating to develop the technology that is required for automated telephone-based multilingual dialogue systems. Such systems require the computer speech application to have access to the telephone network.

Commercially available telephony interfaces are generally expensive, inflexible and platform dependant. In particular, it has proved very difficult to integrate such a telephony interface with open-source software and an open platform operating system, such as Linux. Generally, the hardware and software interface to the device is proprietary and the manufactures do not provide documentation on how to develop a device driver to access their hardware. In some cases, the hardware devices are specially designed to be operated under the Microsoft Windows operating system by removing embedded intelligence from the device and shifting it instead to the Windows driver and hence host CPU. This lowers the cost of the hardware device, but places a greater burden on the host CPU, since the telephony interfaces often require real-time priority.

The aim of this research project is to replace a complex and expensive telephony interface device, based on high-speed DSP and application specific voice processors, with a simpler, microcontroller-based device that can provide adequate functionality to speech recognition applications. The device must provide at least two telephony channels and must be hardware and software portable between different computers and operating systems.

Section II investigates the requirements of a telephony interface to be used by an Automatic Speech Recognition System. Section III discusses why the Universal Serial Bus was chosen as a communication interface, as well as the basic elements of a USB I/O device. An overview of the system is presented in section IV, and the hardware and software design is discussed in section V and VI respectively. Initial tests are indicated in section VII and conclusions are presented in section VIII, together with proposals for future investigations.

II. REQUIREMENTS FOR AN ASR APPLICATION

Automatic Speech Recognition (ASR) is a technology that enables a computer telephony system to recognize a user's spoken words via a telephone connection. The ASR application would typically first prompt the user with prerecorded or synthesised speech. A speech recognizer then

listens for a user utterance. If an utterance is detected, it assumes that it was a reply from the user, and the application will attempt to match this to a vocabulary of known words and sentences in order to determine which words were spoken by the caller.

A telephony interface suitable for use by an ASR application would require the following features:

1) The telephony interface needs to exchange speech data between the telephone channel and the speech recognition application at a rate high enough to allow real-time processing of speech data (speech recognition).
2) The telephony interface must be able to store a few seconds of both incoming and outgoing speech data, as the ASR application would not necessarily be able to process speech data immediately.
3) The telephony interface must provide the means to play audio files to the telephone channel and record speech data from the telephone channel.
4) The telephony interface must be able to notify the ASR application if a "barge-in" or "barge-through" condition has occurred. Barge-in functionality allows users to interrupt a system prompt and to speak without waiting for the prompt to finish playing. This allows a more rapid and natural exchange of information between the user and the system, especially for regular users of the voice service. The telephony interface must stop the playback of a prompt when a barge-in has occurred.
5) The telephony interface must provide adequate echo cancellation. Echo cancellation is an essential feature used by speech recognition technologies to avoid confusing echoed traces of an outgoing prompt with incoming user speech.
6) The telephony interface must notify the ASR application of an incoming call and when a call is dropped. It must be able to answer incoming calls, disconnect active calls, dial telephone numbers and transfer calls.

III. The Universal Serial Bus (USB)

A. Why USB?

The Universal Serial Bus (USB) [1] has several advantages over traditional serial or PCI interfaces that make it an attractive communications interface for telephony interface [1], [5], and [6]:

1) *Single Interface:* A single universal interface is provided that can be used by many kinds of devices. The cables are simple and cannot be plugged in incorrectly. The connectors are small and compact in contrast to other connectors.
2) *Automatic configuration:* When a USB device is connected to a powered system, it can be detected and configured automatically.
3) *No settings:* USB peripherals do not have port addresses or interrupt-request (IRQ) lines. This frees hardware resources for use by other devices and reduces setup requirements.
4) *Easy to connect and "hot pluggable":* There is no need to open the computer casing to install the interface. Most computers have at least two USB ports, and more ports can be added. USB devices can be connected and disconnected as and when needed.
5) *No power supply required:* The USB bus provides +5V and ground power lines. A device that requires up to 500mA can draw all its power from the bus, instead of requiring its own power supply.
6) *Speed:* USB supports three bus speeds: high speed (480 Mbps), full speed (12Mbps) and low speed (1.5Mbps). Every USB-capable computer supports low and full speed. High speed was added in the version 2.0 USB specification. Low speed devices are cheaper as the cables do not require shielding.
7) *Automatic error checking:* The developer does not have to provide error checking algorithms to ensure that data is correctly transmitted and received. This is done by the USB host-controller hardware.
8) *Flexibility:* The USB protocol defines a number of data transfer modes which make it very flexible and suitable for different kinds of applications.

Since USB is a standardised hardware and communications protocol definition, it is platform independent. Hence a USB device can be used without modification under any operating system that supports the USB interface and protocol, for example, Microsoft Windows and Linux. The copyright of the USB 2.0 specification is jointly held by seven corporations (Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC and Philips). They have agreed to make the specification available without charge and founded a non-profit organisation, The USB Implementers Forum (www.usb.org).

B. The USB Device

A USB device requires the basic elements shown in the diagram of figure 1 [5].

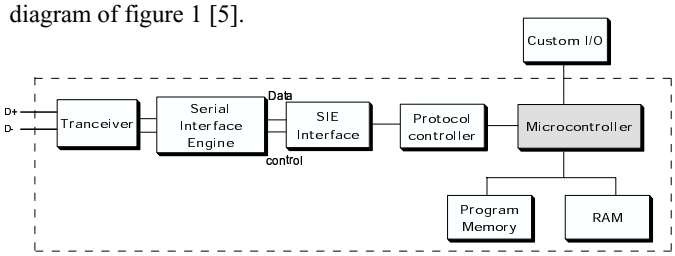

Figure 1: Block diagram of a USB I/O device.

The *USB transceiver* must translate the electrical characteristics of the bus, which uses differential, bidirectional signalling, to the TTL/CMOS voltage levels of the *Serial Interface Engine (SIE)*. The *SIE* receives bits from the USB transceiver, performs error-checking and provides valid bytes to the *SIE interface*. Similarly, bytes are received from the *SIE interface* and transmitted serially onto the USB bus. The *SIE interface* can perform error correction before passing the data to the *protocol controller*. The *protocol controller* handles error conditions, responds to USB events such as the USB handshake protocol and formats incoming and outgoing data to be compatible with the USB packet protocol. The protocol controller is often implemented with a microcontroller or DSP.

Together, these components handle the USB

communications. But, to design a functional USB device, some input and output are needed, together with a microcontroller, microprocessor or DSP to control the flow of input and output signals. Fortunately, these components can be integrated and most vendors include these components on a single chip, called a *USB controller*. The microcontroller would most likely require some RAM and/or ROM to store temporary data and the program code that runs on the microcontroller (*firmware*).

For the purposes of this project, a USB controller that includes a familiar general-purpose CPU, such as the 8051, has been chosen. Cypress' EZ-USB microcontroller family [13] is notable because it supports a different and flexible approach to storing firmware. Instead of storing the firmware on-chip in non-volatile memory, it is stored on the PC host, and downloaded to the USB controller via the USB cable on each attachment. This makes it very easy to update the firmware, since there is no need to replace the chip or use a special programmer. The disadvantage is an increased driver complexity on the PC host and a longer enumeration time. However, once the firmware development is complete, the program code can be stored on an on-board EEPROM.

To save development time, the firmware has been written in the high level language C, instead of assembly language. SDCC [2] (Small Device C Compiler), an ANSI-C compiler designed for 8051-based microprocessors, has been used to compile the firmware for the EZ-USB FX microcontroller. The entire source code for this compiler is distributed under GPL (GNU Public Licence).

IV. System Overview

A platform-independent Application Programming Interface (API) has been developed for speech recognition applications to interface with the hardware telephony interface device. In turn, this API uses functions provided by the *LibUSB* [7] library to interface with the operating system and its host controller driver (figure 2). The *LibUSB* library implements a generic USB driver that provides user-space application access to USB devices.

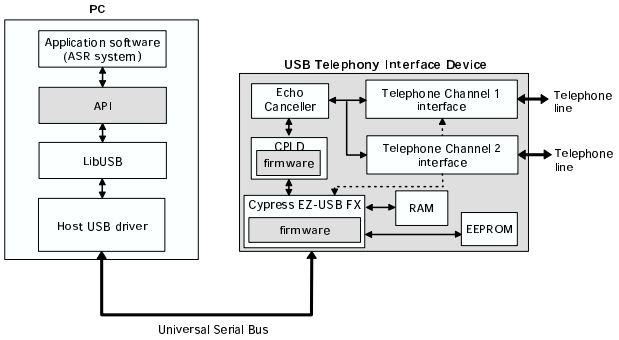

Figure 2: Block diagram of system hardware and software components.

The telephony interface hardware includes a Cypress EZ-USB FX microcontroller for interfacing with the USB bus, as well as devices known as *Direct Access Arrangements (DAAs)* to provide access to the Public Switched Telephone Network (PSTN). Other hardware components include an echo canceller, a Complex Programmable Logic Device (CPLD), RAM, EEPROM, overvoltage protection and power regulation circuitry. These are described in greater detail in the next section.

V. Hardware Design

The integration of the hardware components is shown in figure 3 and discussed in the remainder of this section.

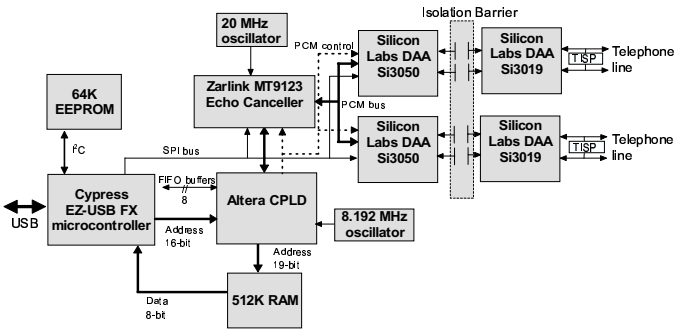

Figure 3: Conceptual overview of the hardware telephony interface.

A. Direct Access Arrangements

Pre-packaged circuits, called Direct Access Arrangements (DAAs) provide interfacing to the PSTN. Such devices are also used in modems, PBX systems and computer telephony applications. They are hybrid circuits or modules that contain many components in a single package. A Silicon Laboratories DAA [3] (Si3050) that meets ITU[1] and ETSI[2] specifications, was chosen as the DAA for use in this project, because it eliminates the need for an analogue front end (AFE), isolation transformer, relays, optocouplers and a 2-to 4-wire hybrid. All of these components are included in two integrated circuits, the *Si3050* (system-side device) and the *Si3018/3019* (line-side device).

The DAA provides two digital interfaces to the microcontroller, a control interface (SPI interface) and a PCM bus for transmitting and receiving of telephony data. The DAA also contains a hybrid network (2-to-4 wire converter). Since both transmit and receive signals are on the same telephone line pair at the same time (full-duplex), a mechanism for the removal of the transmitted signal from the USB device's receive path is required. The attenuation from the transmit path to the receive path is known as the transhybrid loss, and it is desirable to have this loss as high as possible. Unfortunately, as voice signals are transmitted from the four-wire to the two-wire portion of the network, some of the energy in the four-wire section is reflected

[1] International Telecommunications Union (ITU), a United Nations organisation responsible for coordinating global telecommunications activities

[2] European Telecommunications Standards Institute (ETSI) is a standardisation organisation of the telecommunications industry.

(because of an impedance mismatch at the hybrid circuit), resulting in echoed speech. The actual amount of signal that is reflected depends on how well the balance circuit of the hybrid matches the two-wire line. Additional echo-cancellation circuitry can further reduce the echo.

The DAA interface includes a codec (coder/decoder) that uses A-law or μ-law companding. By coding the telephony data, redundant data is discarded. This conserves bandwidth, as less data is transmitted on the USB bus. The signal is then reconstituted on the receiving end (host PC).

The telephony interface must provide high-voltage isolation of the USB device circuitry (digital) from the telephone network (analogue). This is important, as the voltages on the telephone network are high in comparison with the voltages in the digital circuitry of the USB device. The DAA uses a high-voltage capacitor for the communication link across an isolation barrier, where the isolation barrier is a physical separation of the analogue and digital traces on the Printed Circuit Board (PCB). Silicon Laboratories patented this technique as the *ISOcap* technology. It modulates the analogue data with a high-frequency carrier (2 MHz) and passes it across the capacitors to a receiver on the other side of the DAA. An additional path is provided for control and status data, using another capacitor. This capacitive-isolation approach saves board space and makes PSTN integration easy. The disadvantage is that problems with Electromagnetic Interference (EMI) can occur.

Multiple DAAs may be connected in a daisy-chain configuration to provide access to multiple telephony channels.

B. Echo Cancellation

Echoes have many sources, but in PSTN networks, the primary source of echo is hybrid echo. Hybrid echo occurs because the impedance mismatch between the two-wire local loop and the four-wire PSTN network causes a reflection of the outgoing signal. “Hybrids” are used to join the two-wire sections with the four-wire sections, as shown in figure 4.

The Echo Return Loss (ERL) between the transmit and receive paths of the DAA was measured at 22B for a test scenario. According to studies [4], the echo signal will be negligible when the ERL is approximately 55dB or more.

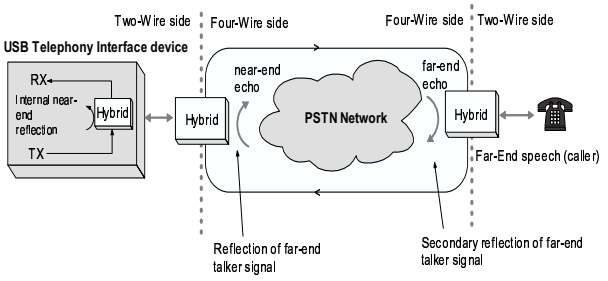

Figure 4: Echoes in the PSTN network (adapted from [16]).

An echo canceller would typically provide an additional 30dB to 40dB of echo attenuation. Zarlink [4] provides a dual channel echo canceller that provides echo cancellation for a tail length [16] of 64 milliseconds. The echo canceller is based on an adaptive FIR filter (figure 5) that subtracts the estimated echo ($\hat{r}(i)$) from the incoming near-end signal ($x(i)$). It uses a convergence algorithm to continuously adapt the filter coefficients to minimize the cancellation error ($e(i)$) when no near-end signal ($x(i)$) is present.

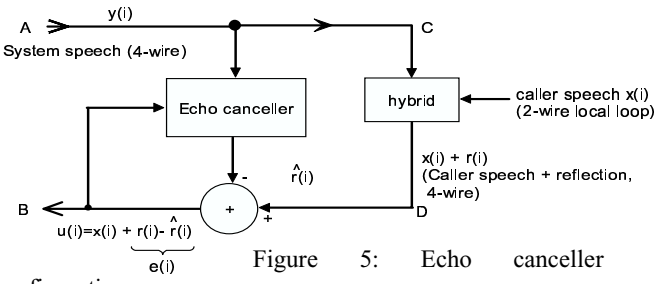

Figure 5: Echo canceller configuration.

The Zarlink MT9123 echo canceller has double-talk detector with a programmable double-talk-detection threshold. Filter adaptation is difficult when double-talk occurs, therefore the echo canceller will stop adaptation during a double-talk condition. A Non-Linear Processor (NLP) removes the residual noise by muting the signal that falls below a certain threshold. Activation of the NLP results in an additional attenuation of the received signal. To prevent a perceived decrease in background noise due to the activation of the NLP, comfort noise injection is performed to keep the perceived noise level constant.

C. Altera Complex Logic Programmable Device (CPLD)

The Silicon Laboratories DAAs requires a 1.024 MHz PCM bus clock signal and an 8 KHz frame signal. These signals are generated by an Altera Complex Logic Programmable Device (CPLD) [8]. The Zarlink echo canceller requires enable strobes to define the channel timeslots to use for PCM data transfers. These strobes are also generated by the CPLD.

The EZ-USB FX microcontroller has no dedicated ports available to interface with the PCM transmit and receive signals. Hence we employ the CPLD to convert the serial PCM stream to bytes and vice versa, which can be read and written to the parallel slave FIFO buffers of the microcontroller. The FIFO buffers are slave in the sense that their read, write and output enable signals may be supplied by external logic (in this case the CPLD). The conversion between bytes and the serial PCM streams is performed by using two shift registers in the CPLD firmware design.

The other task of the CPLD is to perform bank switching of the 512K external RAM, as the microcontroller can only access one 64K bank at a time (16-bit address bus). Additional banks are required for temporary storage of telephony data.

The CPLD design has been carried out by creating VHDL modules and by implementing existing Altera megafunctions.

D. EZ-USB FX microcontroller

The EZ-USB FX microcontroller [13] is a general purpose 48MHz microcontroller, with an enhanced 8051 core that uses four clock cycles per instruction cycle. The microcontroller has a combined 8K internal memory, but can be expanded by adding 64K of external RAM. USB data for bulk and interrupt transfers [1] are stored in fourteen 64-byte endpoint buffers, and can be accessed via registers. The microcontroller has two 64-byte buffers for incoming FIFO data, and two 64-byte buffers for outgoing FIFO data, which can also be accessed via registers in RAM.

The EZ-USB FX initialises and enumerates as a "*default USB device"* [14] when connected to the USB bus. The processor core contains firmware instructions that are able to download new firmware from the host PC. Once the firmware is downloaded, the CPU is reset; the device is "*renumerated"* [14] and the 8051 starts executing the new firmware. The core can only download firmware to internal RAM. Since the firmware that is developed for this project is larger than the internal RAM size of the controller, external RAM must be utilised for code RAM. To download firmware code from the host PC to internal and external RAM of the microcontroller, a special bootloader was developed. The bootloader is downloaded to the microcontroller's internal RAM, from where it writes the firmware code to external RAM. Firmware code downloading to external RAM is thus a three-stage process, but once development is complete, the firmware can be stored in an on-board EEPROM, and the bootloader will no longer be needed.

The EZ-USB FX does not include a dedicated SPI port for communication with the echo canceller and DAAs. A method commonly referred to as "bit banging" is used to create a software-based SPI port. This method uses general-purpose I/O lines to emulate a serial port.

Data is transferred to and from the USB endpoint buffers and the slave FIFO buffers by first storing it in a temporary RAM buffer. The EZ-USB FX incorporates a Direct Memory Access (DMA) engine that transfers data between internal and external RAM without 8051 intervention. Using the DMA, data can be transferred very quickly between different RAM locations (as fast as one byte per cycle of a 48MHz clock).

During an active telephone call, voice-activity detection is performed to determine if the user is speaking. Voice-activity detection is required to determine if a double-talk condition has occurred or to start the recording of speech if sufficient speech energy is present in the incoming signal. The echo canceller's double-talk flag [4] indicates when the input signal is greater than the expected return echo level. The microcontroller firmware reads the double-talk flag, and if it is active, the energy within a window period is calculated. If the energy present in the window period is above a minimum threshold for speech, a barge-in condition is indicated. The signal path for each telephone call will differ, and therefore the noise present in the incoming signal might differ between calls. The minimum speech threshold value is adapted for each telephone call to account for differing channel conditions. To determine the threshold value, a measurement is taken at the beginning of each telephone call, when no user speech is expected. This gives an indication of the noise present in the telephone channel upon which a speech threshold value can be calculated.

To summarise, the firmware running on the EZ-USB FX microcontroller performs the following functions:

1) Sets up the I/O pins and 8051 interrupts that are allowed.
2) Handles standard USB device, interface and endpoint requests [1].
3) Initialises and controls the operation of the DAAs and echo canceller.
4) Handles commands from the API, such as requests for the size of the hardware buffers and channel status, and commands such as dialing numbers, answering and disconnecting active calls etc.
5) Detects if a DAA is receiving a ringing signal, if it is off-hook or if the line is occupied by another off-hook telephone.
6) Moves data between the FIFO buffers, RAM and endpoint buffers to exchange data between the API and the telephone channel during an active call.
7) Perform voice-activity detection (VAD) to determine when the user is speaking (for recording purposes and to detect a barge-in condition).

E. Other hardware components

A 64K serial EEPROM [15] is connected to the I^2C port of the EZ-USB FX. This EEPROM is used to store the final firmware code that will run on the microcontroller.

The line-side DAA connects directly to the telephone line without the need of an isolation transformer. Voltage limiting is thus required to prevent damage from the line transients caused by lightning and power line crosses. A *Totally Integrated Surge Protector* (TISP) [9] was used for this purpose.

Noise transients on a USB cable can cause damage to the USB device if they are of sufficient duration or magnitude. To provide additional electrostatic discharge (ESD) protection to the EZ-USB FX microcontroller, a transient voltage suppressor is connected to the two data lines (*D+, D-*) of the USB bus. A voltage suppressor, SN75240 [10] from Texas Instruments was selected for this purpose.

The majority of the hardware components in this design require a 3.3V power supply. A 5V power supply is delivered by the USB bus to the device, where a voltage regulator [11] provides the 3V supply. The Zarlink Echo Canceller is a 5V CMOS device and some of the device's inputs are not compatible with 3V TTL logic levels. To translate the signals to the echo canceller from 3V TTL logic levels to 5V CMOS logic, a Voltage Translator [12] is used.

VI. Software Design

A. LibUSB

LibUSB [7] (http://libusb.sourceforge.net) is a generic USB driver and open-source library that provides user level access to USB devices. It supports Linux, FreeBSD, NetBSD, OpenBSD, Darwin and MacOS. LibUSB-win32 (http://libusb-win32.sourceforge.net) is a ported version of LibUSB to the Windows operating system, but its API remains the same.

The LibUSB driver fits into the layered driver architectures of the Linux and Windows operating systems, and communicates directly with the host controller driver. The LibUSB API has functions to search the busses for a device, to initialise, open and close a device and to perform bulk, control and interrupt transfers [1] to and from a device endpoint. The USB Telephony Interface Device's API uses these functions to communicate with the device.

B. Application Programming Interface (API)

The API for the Telephony Interface must remain platform- independent, therefore programming was done in ANSI C, and only ANSI C functions and libraries were used. The API for the device provides functions (via the LibUSB driver) to perform the following:

1) Search for the device on the USB bus, open, close, initialise the device.
2) Create buffers in PC RAM where incoming and outgoing telephony data can be stored.
3) Record incoming telephony data to a file or pass it to a speech recognition application.
4) Open an audio file and send the audio data to the telephone channel.
6) Set the behaviour of the voice-activity detector.
7) Set parameters of the device, such as maximum buffer size.
8) Answer an incoming call, disconnect an active call or make a new call by dialing a number.

VII. Initial Tests

The DAAs, CPLD, microcontroller and echo canceller have been tested individually. Currently, the bank switching scheme (memory access) are being implemented and tested for the final prototype. Further testing will include the integration of the system into a practical speech recognition application.

VIII. Conclusion

Although the EZ-USB FX microcontroller has limited capabilities, it successfully demonstrated the feasibility of designing a telephony interface device with a general-purpose CPU, low cost components and non-proprietary software tools and libraries.

In order for the microcontroller to provide at least two telephony channels, the firmware code had to be optimised extensively. If more telephony channels are required, a secondary CPU will be required. Alternatively, a more powerful microprocessor with USB capabilities (such as Freescale's ColdFire processor) could be used. Another future possibility is to adapt the design to provide for ISDN channels, as single-chip ISDN interfaces are becoming available on the market.

References

[1] Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc, Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V, "Universal Serial Bus Specification, Revision 2.0" , April 27, 2000

[2] Sandeep Dutta, "SDCC Compiler User Guide, SDCC 2.4.0", February 24, 2004. http://sdcc.sourceforge.net/

[3] Silicon Laboratories, "Si3050 Global Voice/Data Direct Access Arrangement", Rev 1.0, 2003.

[4] Zarlink Semiconductor "CMOS MT9123 Dual Voice Echo Canceller Data Sheet", Issue 1, October 1996.

[5] John Hyde, "USB design by Example, A Practical Guide to Building I/O Devices, Second Edition", *Intel Press*, 2001

[6] Jan Axelson, "USB Complete, Second Edition", *Lakeside Research*, 2001..

[7] Johannes Erdfelt, "LibUSB Developers Guide", http://libusb.sourceforge.net/doc

[8] Altera, "Max 7000 Programmable Logic Device Family", July 1999, ver. 6.01 .

[9] Power Innovations Limited, UK, "TISP4125F3, TISP4150F3, TISP4180F3 Symmetrical Transient Voltage Suppressors", March 1994, Revised September 1997.

[10] Texas Instruments, "USB Port Transient Suppressors, SN65220/65240/75240", July 2004.

[11] Maxim, "5V/3.3V or Adjustable, Low-Dropout, Low IQ, 500mA Linear Regulators. MAX603/MAX604", September 1994.

[12] Philips Semiconductor, "74LVC425A Octal dual supply translating transceiver; 3-state", 30 March 2004

[13] Cypress Semiconductor, "CY7C6401/603/613 EZ-USB FX USB Microcontroller Data Sheet", 2000

[14] Cypress Semiconductor, "EZ-USB FX Technical Reference Manual", version 1.3, 2000

[15] Microchip Technology Inc., "24AA64/24LC64 64K I^2C serial EEPROM", 2003

[16] Brooktrout Technology, "Echo Cancellation for ASR applications", Keith Byerly, April 2002.

Jaco Müller (main author) was born in Somerset West, South Africa, in 1980. He completed his secondary education in 1998 and obtained his B.Eng (Electric and Electronic) degree from Stellenbosch University in 2003. He is presently studying towards an M.Sc at the same university and part of Telkom's Centre of Excellence (CoE) program.

Thomas Niesler obtained the B.Eng and M.Eng degrees from the University of Stellenbosch in 1991 and 1993 respectively. He subsequently obtained a Ph.D. from the University of Cambridge, England, in 1998. Since November 2000 he has been senior lecturer at the University of Stellenbosch. His research interests lies in speech processing, pattern recognition and statistical modelling. He is also the supervisor of the first author.