Voice Commands for Bluetooth Low Energy Remote Controls

Author(s)	Bougas Pavlos, Iliopoulos Marios
Revision	Rev. A
Release Date	01.04.2018

Dialog Semiconductor

Syggrou Av. 143,	Neue Straße 95
17121, N.Smirni, Athens	73230 Kirchheim/Teck
Greece	Germany
Phone: +30 210 93 10 580	Phone: +49 7021 805-0
Fax: +30 210 93 10 581	Fax: +49 7021 805-100

Introduction

Initially, appliances such as TVs, set-top boxes and air conditioners only needed a handful of controls [1]. An on / off switch, a few selection buttons and two sets of increase / decrease controls were, in most cases, enough to fully control your device.

However, as the number of features that devices support increases, so does the number of commands and configuration options the user has access to. Yet the user still wants to manage them all with a single remote control. To address this, engineers began incorporating more complex user interfaces (UIs). Hierarchical menus appeared on the TV screen, while more and more buttons populated the remote control to allow users to invoke and navigate those menus.

Today, the driving trend is to make devices ‘smart’. A smart device can connect to other devices and to the internet to provide an even larger variety of features and services. Navigating using menus and trying to input large strings by pressing 3 mm x 3 mm keys is impractical, and certainly not a pleasant experience.

In this paper we look at how voice commands can be used to provide a much better user experience. In particular, we examine how they can be implemented via Bluetooth Low Energy (BLE) with the DA14585-based advanced voice remote control reference design from Dialog.

Figure 1 Large QWERTY Remote Control

Using voice as a command interface

Speech is a very powerful and intuitive interface. A short, simple phrase can contain enough information to describe very complex commands. However, capturing a phrase in a noisy environment and deriving a meaningful message, usually as a string, is technically challenging. Fortunately, the issue that created the initial requirement, i.e. the connection of smart devices to the internet, also provides the solution to this complex problem. Devices now have access to cloud computing and can benefit from state-of-the-art voice-to-text recognition engines such as those provided by Nuance Communications, Microsoft, Google, Amazon, and many others. Nowadays, the capabilities of cloud-based speech recognition services are sufficient to provide a very good user experience.

New possibilities arise when feeding the voice to the smart device's virtual assistant. In the last 5 years we have seen a rapid expansion of virtual assistant applications: software agents that perform tasks for an individual. Their capabilities are expanding rapidly with the most widely used being Apple's Siri, Google Assistant, Amazon Alexa and Microsoft Cortana.

Virtual assistants work with text or voice interfaces and use natural language processing to match the text or voice to executable commands. Such applications are already integrated into phones and are rapidly gaining ground in smart speakers, TVs, watches and wearables. Virtual assistants can deliver a wide range of services including providing information, playing music and videos, configuring devices, and even buying items for you.

Why do we even need a remote control?

Solutions that constantly listen for voice commands were developed very early. The device would constantly listen to ambient sounds and search for commands. However, background noise and the user's distance from the microphone made it difficult to identify the message correctly. In addition, the amount of data exchanged between the device and the cloud service was immense, flooding the voice recognition engine with requests, most of them irrelevant. The constant recording of the ambient sound also posed serious security and privacy issues.

A trigger, usually implemented via a button, gesture or recognizable word or phrase, was needed. This solution works well where the user is close to the device, such as with smartphones. But it is much more difficult to correctly identify the trigger and provide a good user experience in smart TVs, set-top boxes and other applications where the user is far from the device. The microphone needs to be close to the user, and the remote control is already there. So, what is more natural than to embed a microphone in the remote?

Quantifying speech recognition requirements

Put simply, the challenge for voice command features can be stated as: "Capture an 'adequate' quality voice recording, send it to the speech recognition engine and then process the text result to derive the user's command". This phrase contains two basic sets of requirements. The first is the need for a trigger. In fact, two triggers are required: the first signals the beginning of the command and the second marks its end.

The second set of requirements relates to the audio signal itself. The voice recording should be encoded in a format suitable for the engine to process and have 'adequate' quality. But what is 'adequate' quality. The Android Compatibility Definition Document [2] describes some ideas on metrics for the quality of audio capture.

The frequency response should be almost flat (+/- 3 dB) across the voice spectrum from 100 Hz to 4000 Hz. This is a well-known specification describing a narrow band voice signal. Regarding the level of the signal generated by the microphone, the Android Compatibility Guide defines a single point in the sound power level-to-RMS diagram and a range over which the sound power level should be tracked linearly.

A sound with sound pressure level (SPL) of 90 dB at 1 kHz should yield an RMS of 2500 for a 16-bit PCM signal. This is almost 10% of the overall amplitude range of a 16-bit signed signal. To get a feeling for the SPL scale, a TV at normal level or a typical human conversation will generate a 60 dB SPL at a distance of 1 m. In comparison, a diesel truck generates a 90 dB SPL at a distance of 10 m.

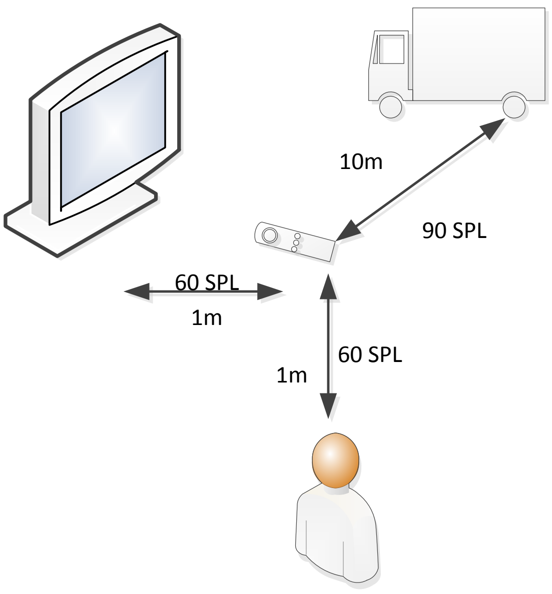

Figure 2 Example of Sound Pressure Level scale

The SPL decreases by -20 dB as the distance from the source increases by a factor of 10. It is easy to imagine how the signal-to-noise ratio varies with the distance from the microphone. Consider a diesel truck 100 m away from you in an open space. If you are 1 m from the microphone, the truck's contribution at the microphone will be 10 dB higher than your voice. If your mouth is 10 cm from the microphone, your voice's contribution will be 10 dB higher than the truck's.

Of course, it is impractical to expect a user to hold the microphone in a precise position and talk at a certain level whenever they want to use voice commands. The voice recognition engine will work for a range of different sound pressure levels given that the PCM amplitude levels can linearly track the changes. A range of at least 30 dB is required. Starting from the

90 dB SPL point, the microphone should track linearly at least from -18 dB to +12 dB; thus between +72 dB SPL and +108 dB SPL. This is equivalent to holding the microphone between 0.8 cm and 25cm from the mouth and talking at a normal intensity.

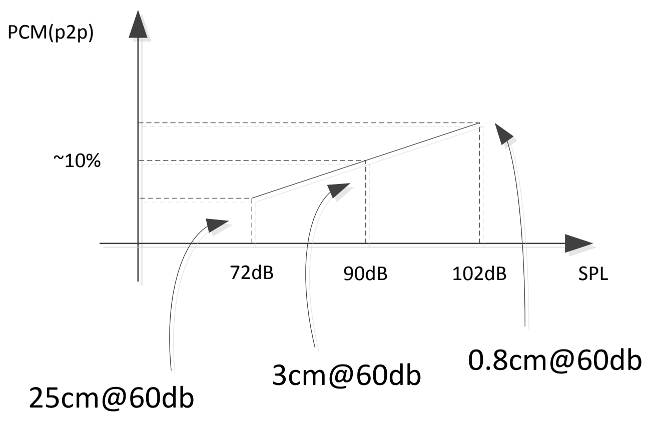

Figure 3 PCM to SPL plot

Voice recognition engines appear to be sensitive to non-linear behavior. Total harmonic distortion should be less than 1% for a 1 kHz sine wave at 90 dB SPL input level at the microphone. Noise reduction processing, automatic gain control, if present must be disabled.

Further requirements

People who have worked with wireless audio protocols in the past are often skeptical of using a data-oriented protocol such as Bluetooth Low Energy to transfer voice commands.

Transferring voice commands is slightly different to transferring real-time audio or human voice such as a telephone conversation. Latency requirements are relaxed as the user does not have to listen back to their voice or maintain a conversation, where fixed and constrained latency is a must. However, loss requirements are strict, because missing pieces of audio may make it impossible for the voice recognition engine to successfully derive the initial user message. The machine can always ask you to repeat the message, but this is not the experience one is looking for.

Architecting a voice command remote control

Now let’s look at the architecture of a voice-command remote control. To do that, we will follow the path of the audio signal through the system. In doing so, we will look at the challenges usually encountered when trying to implement a cost- and power-efficient voice remote control, and their possible solutions.

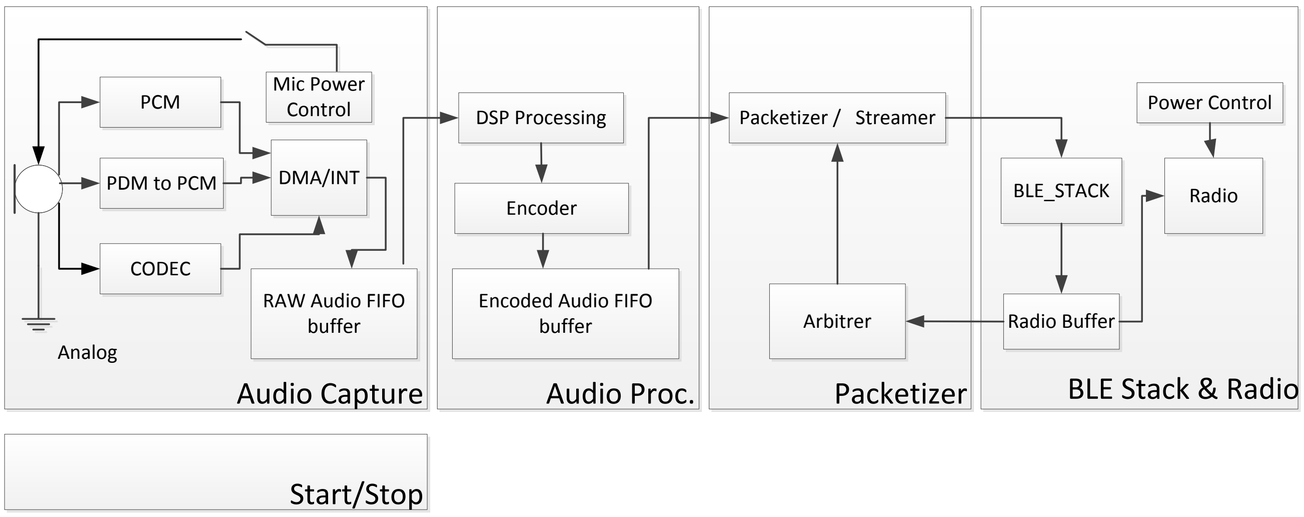

Figure 4 Typical Voice Capturing Signal Path

Everything starts with the audio capturing subsystem. This could be based on an inexpensive analog mic and CODEC circuit, or digital microphones that digitize the samples and deliver them according to a known serial protocol.

For battery powered systems like voice-command remote controls, minimizing power consumption is essential. Thus, gating the power on external components such as the microphone or the external CODEC is highly recommended.

The audio sampling rate must be at least 8 kSamples/s to meet the 4-kHz audio bandwidth requirement. However, 16 kSamples/s with at least 16 bits per sample is a more normal choice. Sampling with 16 bits ensures an adequate range of sound pressure levels, so the captured audio signal will contain enough information for the speech-to-text recognition to work.

Sampling audio involves an interrupt or some form of H/W DMA to acquire the samples and transfer them to a buffer. This buffer is needed to decouple the strictly timed audio sampling from the subsequent audio processing. For low-cost devices, audio processing is handled by the same processor that serves the application and, in some cases, the BLE stack. Thus, the size of the buffer will be dictated by the maximum expected time it will take for the audio processing module to get hold of the CPU, process the audio data and move them onto the next step. Typical times are in the range of a few ms. For a 16-bit, 16-kSamples/s signal, 32 bytes are generated for every ms and a 160-200 bytes buffer is considered adequate to allow processing time in the range of 5 ms.

The audio processing module implements simple audio processing and encodes the audio data to reduce their overall rate. Audio processing includes very basic filtering such as dc-offset removal or bandpass filtering, and possibly a fixed gain step to optimize the audio amplitude. The raw audio data is 256 kbit/s and can be marginally streamed over BLE. To lower the rate and to better utilize the bandwidth, the audio is encoded using known audio compression algorithms. The encoder selection varies from simple, fixed-rate lossy codecs like IMA-ADPCM to complex, processing intensive, fixed or variable rate algorithms such as OPUS. Simple codecs can run on CPUs with a lower MIPS capacity but will produce a lower quality audio stream for the same output bit rate. On the other hand, complex algorithms increase the quality of the encoded stream but require a more expensive and power-hungry CPU.

The encoder's output contains the final payload that needs to be streamed over the air, and an additional buffer is required to help equalize the instantaneous RF data rate with the encoder's mean output rate. The encoder's output rate is equal to the sample audio rate divided by the compression ratio achieved. Below is a table showing typical raw and encoded audio rates supported by Dialog's Voice RCU reference design.

Table 1 Raw and IMADPCM Encoded Audio

Sampling Rate	Raw Audio Rate	Compression Ratio	Encoded Audio Rate
16 kHz	256 kbit/s	1:4	64 kbit/s
16 kHz	256 kbit/s	3:16	48 kbit/s
8 kHz	128 kbit/s	1:4	32 kbit/s
8 kHz	128 kbit/s	3:16	24 kbit/s

BLE is a packet-based protocol, with packets exchanged at specific time instances separated by a periodic connection interval. If an interferer distorts the packets during a rendezvous point, data will be retransmitted in the next one. Every connection interval happens in a different frequency channel. Often an interferer appears constantly in a frequency range, distorting multiple connection events and significantly reducing the bandwidth.

BLE provides a mechanism called channel map update to tackle this problem. The master will detect the frequency range affected and issue a channel map update procedure. Until this happens, the BLE connection might experience a significant drop in the RF data rate. The buffer at the encoder's output should be sized accordingly to be able to sustain this kind of event. Sizing can include ultra-safe approaches, such as the ability to buffer a complete 5 second message which requires 40 kbytes, or not to lose a single byte of data with half the rate for 5 seconds which requires a 20 kbytes buffer. Looking into the available devices in the market, this is a very hard requirement to meet. Most devices have 20 – 40 kbytes of total RAM available for the complete stack and application. This resource cannot be wasted on a single buffer. One thing to note is that the size of the buffer is directly proportional to the encoder output rate and the time we want to ensure that no loss of data will occur.

Various techniques have been proposed which can be used alone or in combination to tackle this problem without having very large buffers. Increasing the RF output power when streaming audio samples to minimize the effects of possible interference; fast fine tuning of the channel map; using more complex encoders to lower the required RF data rate or even dynamically dropping the quality when significant RF bandwidth decreases are detected. Each technique tries to handle the problem at the expense of some other resource such as power, CPU overhead or audio quality instead of using more RAM.

Depending on the encoder algorithm used, the data can be stored either as a continuous stream or as a list of packets of a predefined length. To better utilize the BLE bandwidth, data is best viewed as a stream with no form of packetization from the encoder. The lossless nature of BLE transmission ensures that we can always reconstruct our initial stream on the remote. A number of BLE connection parameters – the connection interval, ATT_MTU size, data channel PDU size and connection event length – influence the selection of the theoretical best size of the BLE packets. At runtime the actual error rate may require further optimization of the packet size. Using large data PDUs may effectively double the effective data rate, since the time used for the response of the remote and the inter-packet gap is eliminated. On the other hand, the transmission time of a single packet grows, increasing the possibility of an error occurring during the transmission of the packet and the cost of retransmission as well. In heavily polluted environments it may be more efficient to use a smaller data PDU size than the maximum negotiated. All these requirements dictate the presence of a smart module that will pull data out of the encoded stream, packetize it according to the connection characteristics and push it into the BLE stack while monitoring the real BLE data rate and the error rate. The packetizer should also have some form of arbitration and flow control so as not to flood the BLE stack and keep enough bandwidth for other application entities that may want to use the BLE for data exchange during an audio transmission to ensure the device will not become unresponsive.

Implementing the host

An application hosted in the device to be controlled will receive the audio data, re-assemble the audio stream and feed it to the decoder to get the initial PCM stream. Depending on the cloud service selected, the data will have to be re-sampled and / or re-encoded to a new format suitable for the service's speech-to-text engine. The audio data can be streamed to the engine as a complete file in one go or in parts as each sample arrives from the remote control. The same applies for the engine's response: the engine can answer once, after receiving notification of the end of the string, or it can answer every time we ask for a result for the audio data we have streamed to that point. Implementing an application with the described functionality is straightforward.

For systems like TVs or set-top boxes, where the manufacturer has full control of the operating system (OS) code, a different integration may be applicable. In such systems, a special driver may help to interface the remote control with the OS audio subsystem as a virtual microphone. The remote control is usually presented as an HID device, which are generally managed by the OS. Thus, nothing has to be developed to connect, bond, discover and manage a remote control. Audio is not a part of the HID specification, but we can extend the HID report map and include vendor specific characteristics to use for the audio data and commands. A custom driver may be hooked up to receive the data traffic issued through the vendor specific characteristics. When audio data is sent, the driver decodes it and pushes it to the OS audio subsystem to feed the virtual microphone. In this approach, all applications, even system applications, can access the audio data streamed, enabling services like “Ok Google!” or “Hey Alexa!”.

Designing a smart voice remote control

Making decisions about a voice remote control design is a multi-parameter problem. Remote control devices are generally mass produced and, thus, cost sensitive. Power consumption is another key parameter. No user wants to change batteries often, especially for features that might be used rarely. Those two requirements usually constrain the overall design. Audio quality plays a significant role in use cases where the command is repeated as an audible confirmation. On the other hand, latency is mainly determined by the cloud voice recognition service rather than the latency in the data transfer and intermediate buffering. There are already techniques where the voice service will start replying before the user finishes their message, providing a feeling of instant response and hiding several seconds of actual latency.

Dialog advanced voice remote control

To enable designers to evaluate the performance of a full-featured voice command remote control, Dialog offers an advanced voice remote control reference design based on a single DA14585 BLE SoC. The design supports not only voice but a range of additional features such as wireless mouse, touch pad, IR and, of course, keys – all of which would be expected in a modern RCU design to give users more natural ways to interact with the end device and navigate through menus and results.

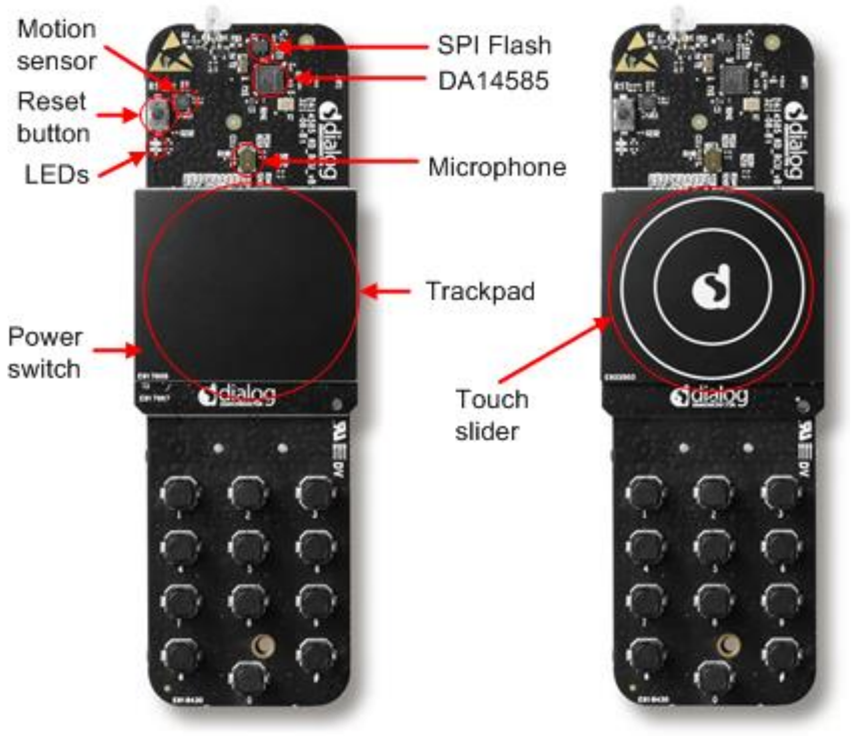

Figure 5 Dialog's Voice Remote Control Development Kit

The second-generation reference design includes significant changes in the overall software architecture to make it more modular and easier to adapt to end-product requirements. All functionality is organized into modules, with platform dependent and independent parts, hooking on drivers built for the DA14585's internal peripherals or external components selected for the Dialog Advanced Voice RCU development board.

The audio path is architected around the principles presented in the previous section.

-

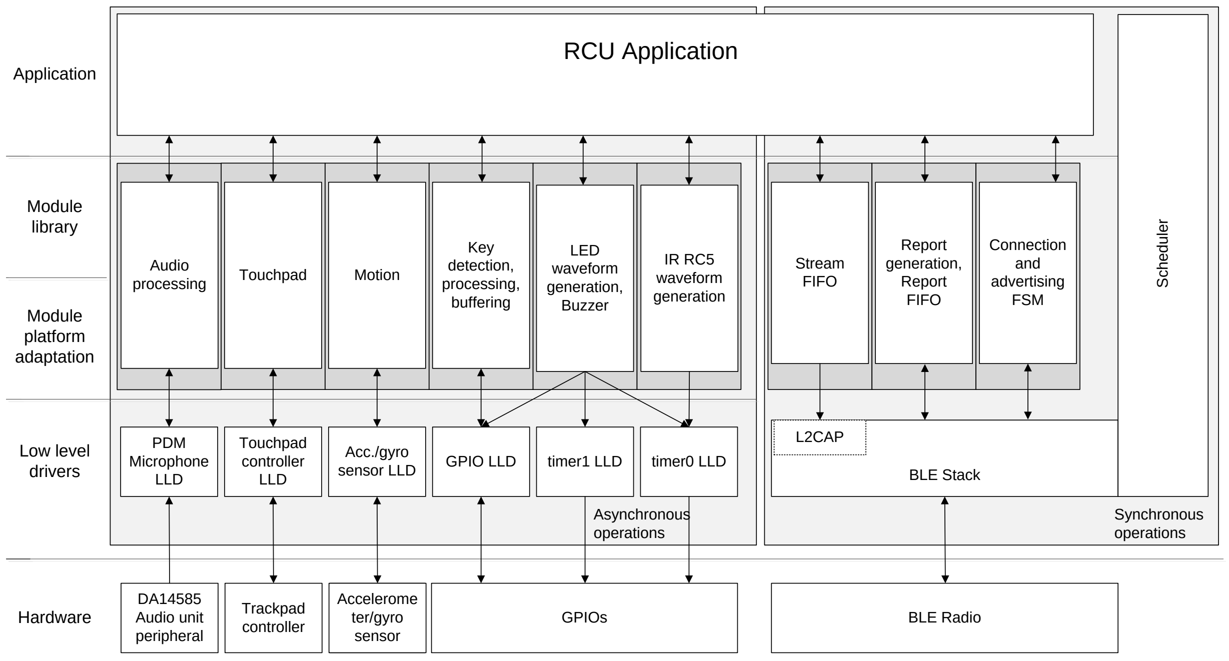

Figure 6 System Architecture of Dialog's Voice Remote Control

To increase audio quality, the design features a digital microphone. For cost-sensitive applications, a PDM microphone is a better fit than an I2S/PCM one. The DA14585's integrated PDM-to-PCM converter transforms high-frequency PDM signals into 24-bit audio PCM samples and delivers them to a memory buffer via a dedicated DMA channel.

The M0 CPU executes the audio processing and streamer modules. From the main loop, a pre-processing block is called to apply a DC blocking and a 24-bit-to-16-bit conversion. A down sampling unit can halve the audio sampling rate from 16 kSamples/s to 8 kSamples/s if required. The result of the audio pre-processing unit is fed to the IMA ADPCM encoder. Depending on the configuration, the encoder produces a 4- or 3-bit sample for every 16-bit audio sample.

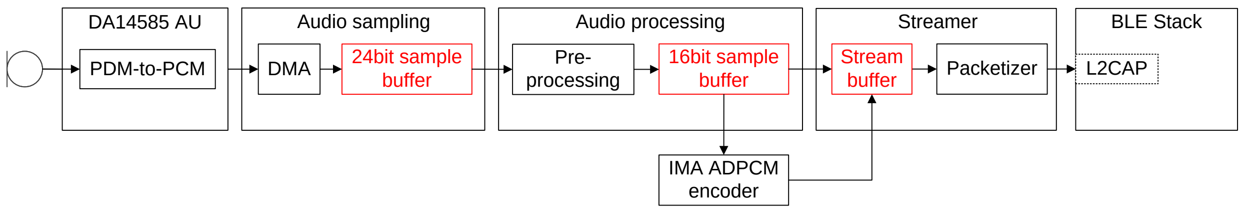

Figure 7 Dialog's Voice Remote Control audio path

As a design choice, the reference design supports an adaptive audio rate mechanism to guard against possible buffer underruns. The sample rate of the audio samples and / or the configuration of the IMA ADPCM encoder will change on the fly, making it possible to switch between different output rates. To support this mechanism, an in-band signaling mechanism has been developed. The encoder's output and the in-band signaling share the same stream buffer.

The packetizer module collects data from the stream buffer and pushes them into the stack efficiently while trying not to overflow the stack. As well as taking connection parameters into account, the packetizer also closely monitors the error rate and instantaneous available bandwidth. This allows it to make decisions on whether to lower or increase the quality when the adaptive audio rate mechanism has been enabled.

All policies, such as the use of a fixed or adaptive audio rate, the way the audio starts and stops, is controlled by the user or the remote device. Individual buffer sizes are all configuration options because, depending on the end application, a different user experience may be suitable each and every time.

Conclusion

The ability to connect devices to the internet combines with modern cloud-based speech recognition service to enable a powerful new user interface – the voice command. Smartphones, smart TVs and set-top boxes already use such commands. The user's experience of speech recognition can be greatly enhanced by integrating low-cost microphones into BLE-connected peripheral devices. Commands collected from remote controls, smart watches, and wearables and routed via a smart device to a speech recognition engine in the cloud can be used to control both the smart device itself and peripheral device connected to it or controlled by a voice assistant.

References

[1] https://en.wikipedia.org/wiki/Remote_control

[2] https://source.android.com/compatibility/android-cdd.pdf

Further reading

[1] https://www.dialog-semiconductor.com/sites/default/files/dsa009abluetoothsmarst-multi-function-remote-controls_0.pdf

[2] https://www.dialog-semiconductor.com/products/da14585-voice-rcu-development-kit