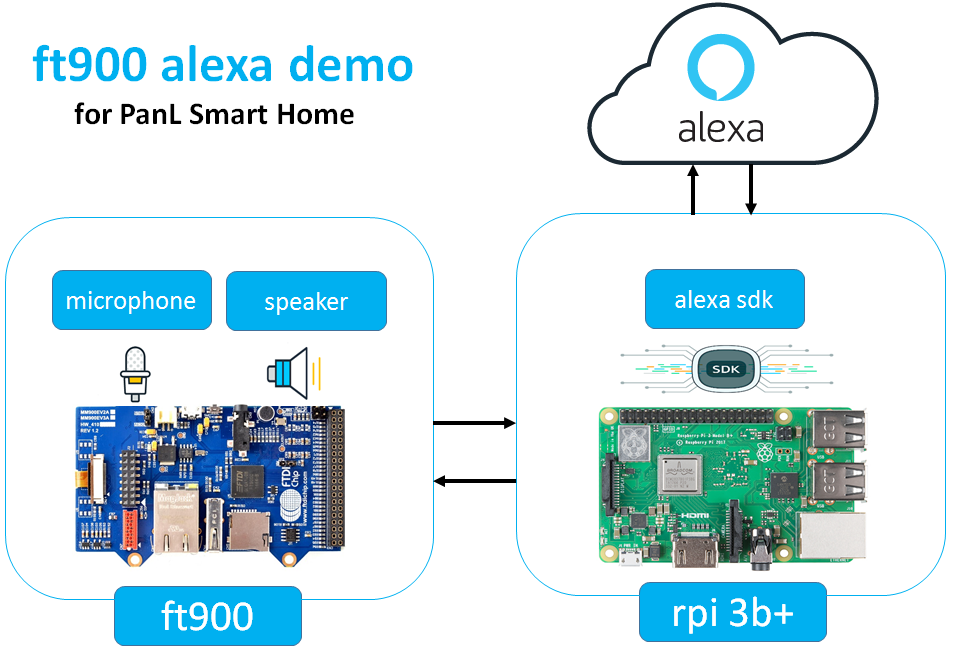
**FT900 Alexa Demo**

**FT900 Alexa Demo**

This PoC application demonstrates using FT900 microcontroller as an Amazon Echo Dot device, where users can issue voice commands to Alexa and hear Alexa’s voice responses generated by Alexa’s complex speech recognition and natural language processing in the cloud. To make this possible, FT900 (Alexa client) communicates with a Raspberry PI 3B+ (Alexa gateway), which relays voice requests and voice responses to and from the Alexa cloud. The RPI runs a customized version of Amazon’s official open-source Alexa Voice Service (AVS) SDK, written in C++.

****

**Github Code:** [**https://github.com/richmondu/FT900/tree/master/Alexa/Amazon%20Alexa%20Client**](https://github.com/richmondu/FT900/tree/master/Alexa/Amazon%20Alexa%20Client)

**PanL Smart Home with built-in Alexa**

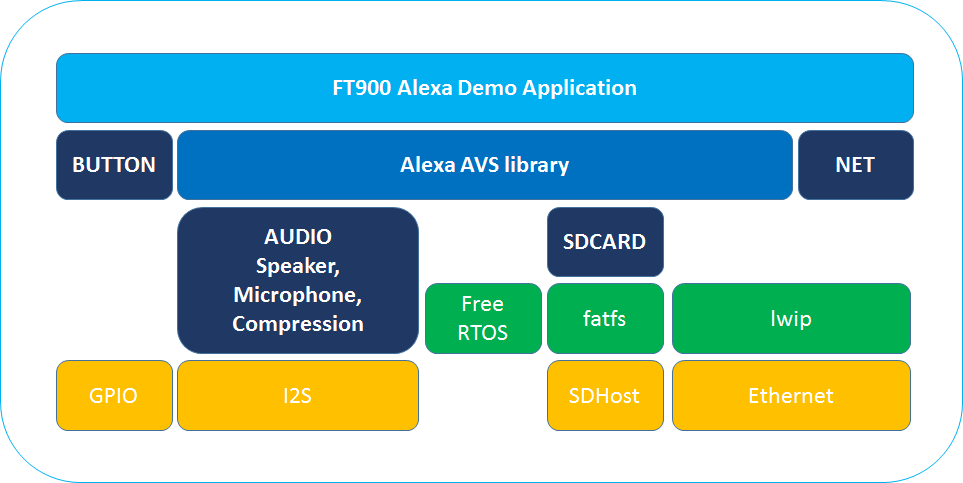
This demo is targeted for FTDI/Bridgetek’s smart home devices, PanL Hub and PanL Display. Having Alexa built-in to PanL products allow customers to talk directly to Alexa via PanL without needing to buy Amazon Echo devices. Furthermore, customers will have access to the built in capabilities of Alexa (like music playback, timers and alarms, package tracking, movie listings, calendar management, and more) including third-party skills developed using the Alexa Skills Kit.

PanL Hub, which runs on RPI, acts as the Alexa gateway while the PanL Display, which runs on FT900 microcontroller, acts as the Alexa client. Customers will be able to use both PanL Hub and PanL Display to issue voice commands and operations to Alexa.

****

**FT900 Alexa Client – Block Diagram**

Below is a block diagram of the Alexa Demo on the FT900 side.

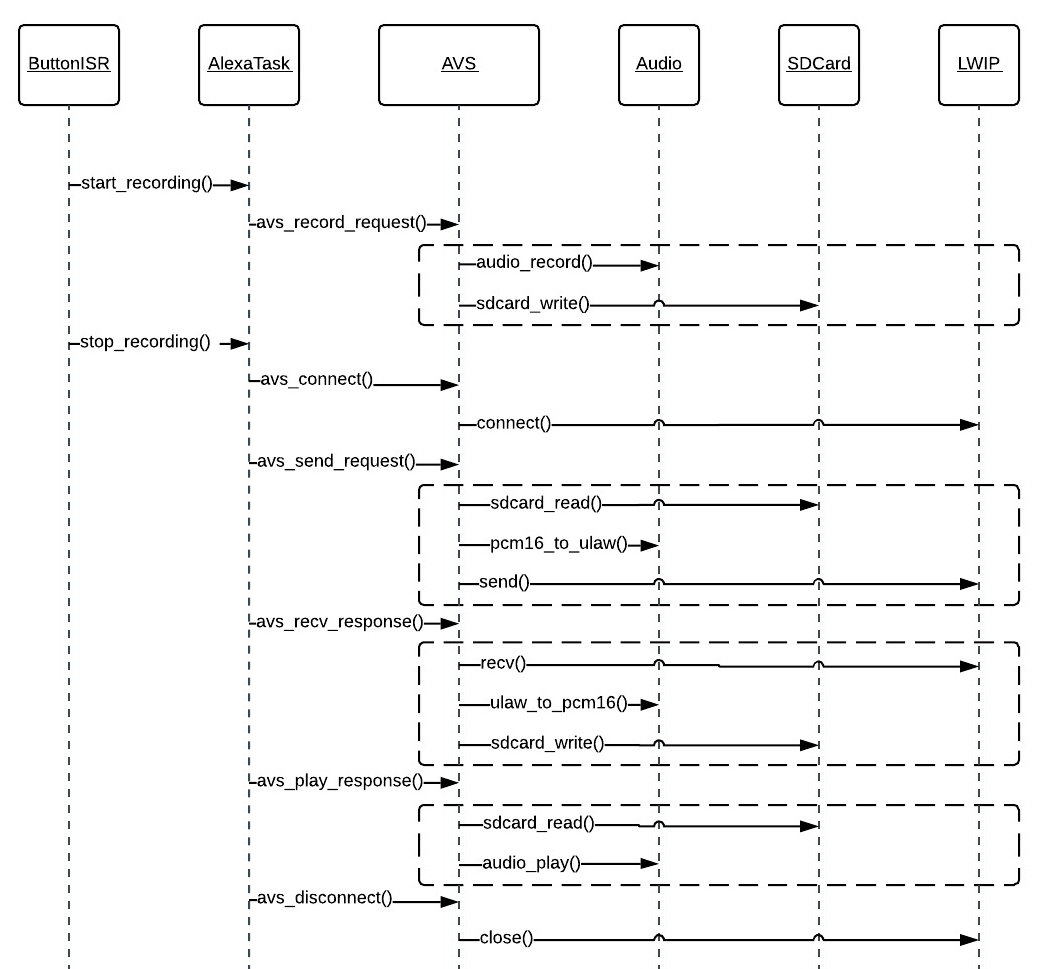


The main component of the Alexa Demo on the FT900 side is the **Alexa AVS library**. The library abstracts the audio, the SD card and the network communication by providing easy to use API interfaces. The Alexa AVS library contains the following functions:

|  |  |
| --- | --- |
| **Functions** | **Descriptions** |
| avs\_connect | Establishes connection to the RPI Alexa Gateway using configurations in avs\_config.h configuration file. |
| avs\_disconnect | Closes connection with RPI Alexa Gateway. |
| avs\_record\_request | Records voice request from microphone and saves it to SD card given the provided filename.  **Audio recorded**: 16-bit PCM, 16KHZ, stereo (2-channels)  **Audio saved**: 16-bit PCM, 16KHZ, mono (1-channel) |
| avs\_send\_request | Sends the voice request to the RPI Alexa Gateway provided the filename of the voice recording in the SD card.  **Audio read**: 16-bit PCM, 16KHZ, mono (1-channel)  **Audio sent**: 8-bit u-law, 16KHZ, mono (1-channel) |
| avs\_recv\_response | Receives the voice response from the RPI Alexa Gateway and saves to SD card given the provided filename.  **Audio received**: 8-bit u-law, 16KHZ, mono (1-channel)  **Audio saved**: 16-bit PCM, 16KHZ, mono (1-channel) |
| avs\_play\_response | Plays the voice response from the SD card to the speaker.  **Audio read**: 16-bit PCM, 16KHZ, mono (1-channel)  **Audio played**: 16-bit PCM, 16KHZ, stereo (2-channels) |

**FT900 Alexa Client – Sequence Diagram**

Below is a sequence diagram of the Alexa Demo on the FT900 side.



Notes:

1. **Wakeword detection** is not supported. To trigger voice recording, user has to press down a button and release it to stop recording, similar to remote control for Amazon Fire Stick TV.
2. Data sent is compressed from **16-bit to 8-bit** using **u-law G711 compression algorithm**. Data received is expanded from 8-bit to 16-bit using u-law G711 expanding algorithm. Compressing the data before transmission reduces the data bandwidth by half.
3. Alexa cloud expects **16-bit, 16KHZ, mono (1-channel)** **PCM** voice recording. For response, Alexa cloud sends out an **MP3** data stream. RPI converts the MP3 data stream to **PCM** before compressing to 8-bit and sending to FT900.
4. Converting stereo to mono data stream is done by averaging the consecutive 16-bit WORDs.

**FT900 Alexa Client – Performance and Sample Execution Log**

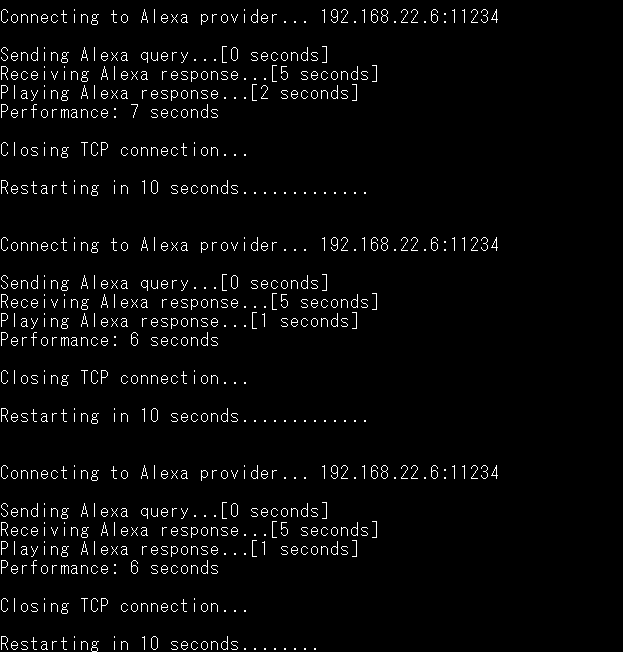
It can be observed in this log below that the round-time on the FT900 side is 6-7 seconds. This measures the time FT900 sends the Alexa request until plays Alexa response. This is for a simple command, “What time is it?”.

It takes 5 seconds to process receiving Alexa response. Receiving Alexa response consists of:

- receiving Alexa request (8-bit)

- expanding it to 16-bit

- saving it to SD card.



**FT900 Alexa Client – Audio Quality**

**A. FT900 Speaker playback**

Playback of Alexa response on connected speaker is very very good. There is no noise or jittering. It is very smooth.

This was made possible by maximizing the FIFO buffer sizes for both SD Host (4KB) and I2S Master (2KB). 4KB data is read from SD card then segmented into 4 1KBs. Each 1KB data is converted from mono (1-channel) to stereo (2-channels). This results to 2KB stereo data which is then written to I2S Master speaker. The process is repeated until the recorded audio file is completely processed.

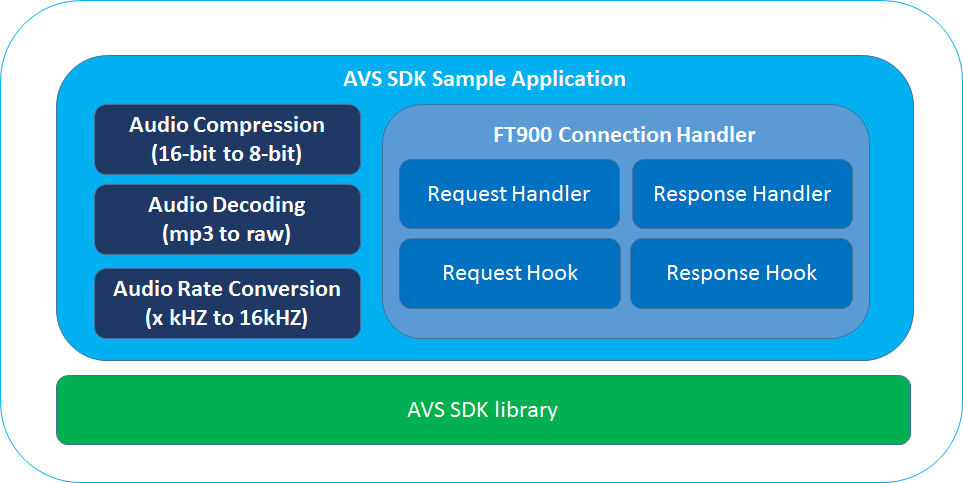
**B. FT900 Microphone recording**

Recording of Alexa request on FT900 microphone is good. Background noise can be heard but voice pops out when user speaks very near to microphone.

**RPI Alexa Gateway – Block Diagram**

Amazon provides an official **Alexa Voice Service (AVS) SDK**, (written in C++) <https://github.com/alexa/avs-device-sdk> (Note that the Java, Python and NodeJS SDKs are already obsoleted and are no longer publicly available.) The version I am using is **AVS SDK is v1.11.0**, (**12-19-2018)**. Instructions to install on the AVS SDK on RPI can also be found on the github link above.

Below is a block diagram of the Alexa Demo on the RPI-side.

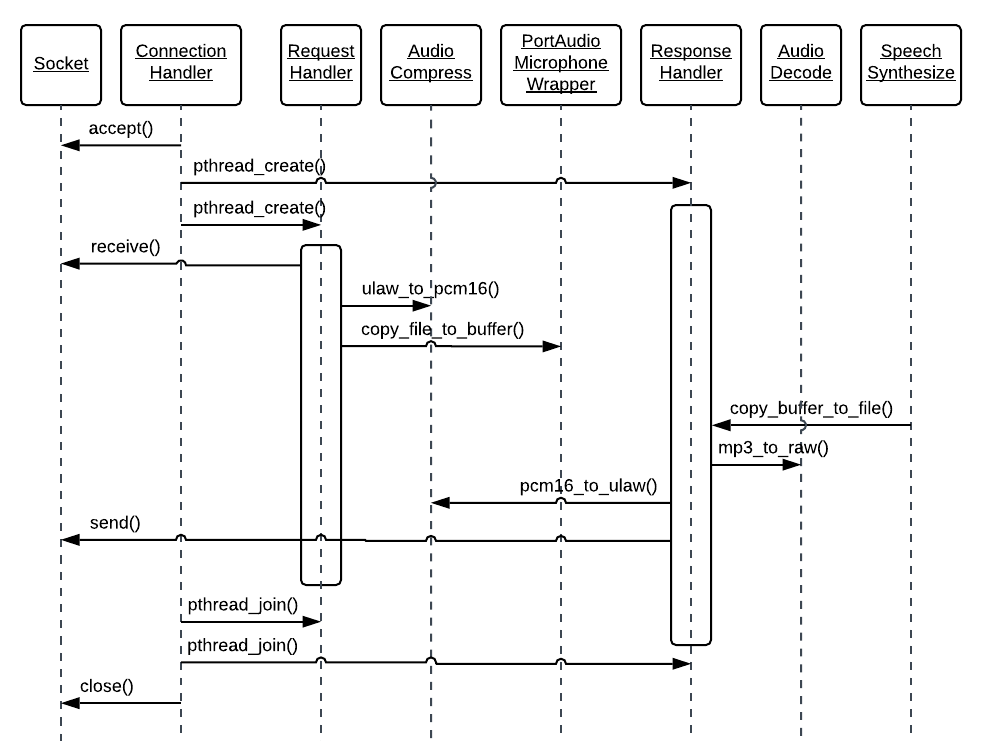


Below are the Alexa Demo modules that were implemented and modified on the RPI-side.

|  |  |  |
| --- | --- | --- |
| **Files Implemented** | | **Descriptions** |
| FT900ConnectionHandler.cpp | | Handles connection with FT900 and creating threads for both FT900RequestHandler and FT900ResponseHandler. |
| FT900RequestHandler.cpp | | Handles processing of Alexa request from FT900. |
| FT900RequestHook.cpp | | Handles hook of request to PortAudioMicrophoneWrapper. |
| FT900ResponseHandler.cpp | | Handles processing of Alexa response to FT900. |
| FT900ResponseHook.cpp | | Handles hook of response from Speech Synthesizer of library. |
| FT900AudioCompression.cpp | | Handles ulaw 8-bit compression and expansion. |
| FT900AudioCompressionHelper.cpp | | Helper function for FT900AudioCompression.cpp. |
| FT900AudioDecoding.cpp | | Handles decoding of mp3 response to raw pcm. |
| FT900AudioRateConversion.cpp | | Handles conversion from x KHZ to 16 KHZ. |
| **Files Modified** | **Descriptions** | |
| SampleApplication.cpp | Initialize FT900ConnectionHandler in separate thread. | |
| PortAudioMicrophoneWrapper.cpp | Feed in request data to microphone datastream. | |
| SpeechSynthesizer.cpp | Call the callback function when response is received. | |
| UserInputManager.cpp | For testing of pre-recorded audio requests. | |
| MediaPlayer.cpp | Fix audiosink issue. | |
| DefaultClient.h | Provide access to Speech Synthesizer handle. | |

**RPI Alexa Gateway – Sequence Diagram**

Below is a sequence diagram of the Alexa Demo on the RPI-side.



Notes:

1. The primary modifications for the AVS SDK application are contained in the following classes:

**PortAudioMicrophoneWrapper**: PortAudioCallback() contains the data stream for Alexa request

**SpeechSynthesizer**: startPlaying() contains the data stream for Alexa response

1. Audio properties for Alexa Demo on the RPI-side:

**Audio received** (from FT900): 8-bit u-law, 16KHZ, mono (1-channel)

**Audio sent** (to Alexa cloud): 16-bit PCM, 16KHZ, mono (1-channel)

**Audio received** (from Alexa cloud): MP3

**Audio sent** (to FT900): 8-bit u-law, 16KHZ, mono (1-channel)

1. **SOX** utility is used to decode MP3 data stream to raw PCM16 data stream.
2. **G711 u-law** lossless companding (compression/expanding) algorithm is used to convert data stream from 16-bit to 8-bit and vice versa, without affecting audio quality. Compressing the data before transmission reduces the data bandwidth usage by half.

**RPI Alexa Gateway – Detailed Explanation**

1. A ConnectionHandler thread is initialized in the main function of the AVS SDK SampleApplication.

2. ConnectionHandler thread waits for an FT900 connection.

3. Once an FT900 connected, it initializes 2 threads, FT900RequestHandler & FT900ResponseHandler.

4. FT900RequestHandler handles the processing of Alexa requests from FT900.

5. FT900ResponseHandler handles the processing of Alexa responses to FT900.

6. Only 1 FT900 can connect at a time.

7. FT900RequestHandler receives Alexa request (8-bit compressed using ulaw algorithm) from FT900.

8. FT900RequestHandler decompresses/expands the Alexa request from 8-bit to 16-bit.

9. FT900RequestHandler copies the data stream to the data buffer in PortAudioMicrophoneWrapper.

10. The Alexa request is then sent to the cloud and receives the Alexa response in MP3 format.

11. SpeechSynthesizer copies the data stream to an MP3 file.

12. SpeechSynthesizer converts the Alexa response from MP3 format to raw PCM format.

13. FT900RequestHandler compresses the Alexa response to 8-bit from 16-bit.

14. FT900RequestHandler sends Alexa response (8-bit compressed using ulaw algorithm) to FT900.

14. ConnectionHandler thread waits FT900RequestHandler and FT900ResponseHandler to terminate.

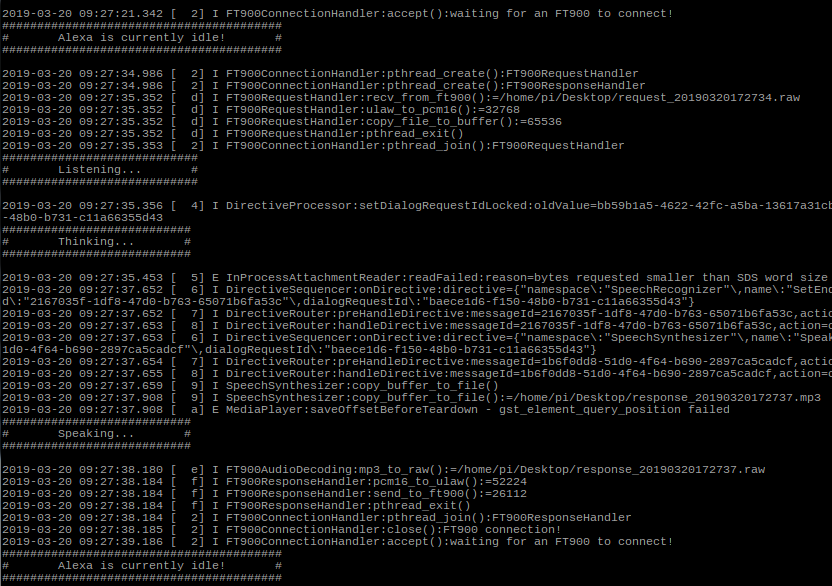
15. ConnectionHandler then closes the socket for the FT900 connection.

**RPI Alexa Gateway – Performance and Sample Execution Log**

It can be observed in this log below that the round-time on the RPI side is about 3.199 seconds. This measures the time RPI receives FT900 connection until it closes the connection. This is for a simple command, “What time is it?”.

09:27:34.986 FT900ConnectionHandler::pthread\_create():FT900RequestHandler

09:27:38:185 FT900ConnectionHandler::close():FT900 connection!



Note that the logs more or less correspond to the sequence diagram. This is made as such so that it would be easy to understand the code on a high level.

**PanL Smart Home Adoption analysis**

### A. RS485 small bandwidth issue

The demo currently uses Ethernet for communication between RPI and FT900. For PanL, where communication medium is RS485, bandwidth is smaller, about 92KBps only. This is one of the major concerns for adoption of the solution to PanL Smart Home Solution. Below is an analysis that answers this concern.

The size of an 8-bit 16khz response for a simple question "What time is it?" is less than 32kb. This is sent in 2ms. 30720bytes \*1000/2ms=15360000 (14.6 MBps)

To simulate RS485 slowness, I added delay between each send (note that 32kb is sent in chunks as it is compressed in chunks). Results:

1. 20ms delay => response is sent in about 1sec instead of 2ms. (32k bytes/sec) \*causes stutter
2. 13ms delay => response is sent in about 660ms instead of 2ms. (48k bytes/sec)
3. 10ms delay => response is sent in about 500ms instead of 2ms. (64k bytes/sec)
4. 6ms delay => response is sent in about 330ms instead of 2ms. (96k bytes/sec)

The 20ms added delay sometimes causes a stutter for recv\_and\_play\_no\_sdcard option as sender becomes slow. Meaning, using SD card to save response is necessary when sender rate is around 32KBps only.

But since the acceptable computed rate for RS485 is 64KBps (70% of 92KBps), then not using SD card to save response is still OK. If we use SD card to save response, then RS485 is not a even problem.

Note that the demo solution provides both options to save or not to save response to SD card. (Saving response to SD card is 1-2 seconds slower than NOT saving response to SD card.)

### B. CPU usage

### Another concern is the CPU usage consumed by the application on RPI.

### Based on observation, the CPU usage jumps to 20-30% for a split second when processing a request. This is OK. Note that no AI is done on the RPI. The Alexa SDK only forwards the request on the cloud where the AI-generated response is created. So the Alexa application on RPI does NOT and can NOT hog the CPU.

**Setup Guide – RPI**

Prerequisites:

A. RPI

1. RPI 3B+

2. Headphone or speaker

3. USB microphone <https://circuit.rocks/usb-mini-microphone.html>

B. FT900

1. FT900 Rev C board

2. SD card (<= 32gb)

3. Headphone or speaker

4. (Optional) Button <https://circuit.rocks/button-digital> (for GPIO mode)

RPI setup:

Download the latest [RPI Alexa Gateway](https://github.com/richmondu/FT900/tree/master/Alexa/Amazon%20Alexa%20Gateway) code.

A. Install AVS SDK (latest version is AVS SDK 1.12.0 [02-28-2019])

1. Install the original AVS SDK on RPI using the official installation guide.

<https://github.com/alexa/avs-device-sdk/wiki/Raspberry-Pi-Quick-Start-Guide-with-Script>

2. Run and verify everything is working as expected.

Note: press 't' key to issue voice command to Alexa.

B. Integrate AVS SDK modifications (AVS SDK 1.11.0, [12-19-2018])

1. The RPI Alexa Gateway is a customized AVS SDK.

Replace the original avs-device-sdk folder with this modified avs-device-sdk.

2. Install SOX utility

sudo apt-get install sox libsox-fmt-mp3

3. Compile and run.

Note: You should see logs containing 'FT900'.

4. Setup and run FT900.

**Setup Guide – FT900**

FT900 setup:

Download the latest [FT900 Alexa Client](https://github.com/richmondu/FT900/tree/master/Alexa/Amazon%20Alexa%20Client) code.

A. Test Mode (Use a pre-recorded request contained in SD card)

1. Change AVS\_CONFIG\_SERVER\_ADDR in avs\_config.h.

This should contain the IP address of the RPI

2. Copy test/request.raw to SD card.

request.raw is an audio recording of "What time is it?"

3. Compile and run.

This automatically sends the pre-recorded audio, request.raw, to RPI.

B. Normal Mode (User presses a key or button to trigger voice recording)

1. Change AVS\_CONFIG\_SERVER\_ADDR to avs\_config.h.

This should contain the IP address of the RPI

2. Change TEST\_MODE to 0 in main.

3. (Optional) Change USE\_GPIO to 1 in button.c.

(Optional) Connect button to GPIO 31, 5V and GND.

(Optional) Default is UART instead of GPIO.

4. Compile and run.

5. For UART mode, Press 't' to start recording voice command.

For GPIO mode, press the button.

6. For UART mode, press 't' again to stop recording voice command.

For GPIO mode, release the button.

7. After stopping the recording, FT900 will send the voice request to RPI.

**Action Items**

Below are the action items for the Alexa Demo.

1. Support for **wake-word detection in FT900.** (Currently, user has to press down a button to start voice recording.)
2. **Use 8KHz instead of 16KHz** to reduce audio file size by half. (However, note that RPI will have to convert the 8KHz to 16KHz as Alexa cloud requires 16KHz).
3. **Performance/speed optimization**
4. Support for **alarms or notification-based messages**
   1. Currently, only responses triggered by requests are supported.
5. Support for **very long Alexa responses**.
   1. Need to test requests that have very long responses.
6. Support for **queuing Alexa requests from multiple FT900 clients**.
   1. Multiple FT900 can simultaneously send requests to RPI. RPI should queue the requests and only issue a request when a response for previous request is processed.
7. **RPI should not play response on its speaker** when the request is from FT900.
8. **Audio decoding implementation** currently uses bash scripts using **SOX** utility.
   1. Should be replaced with C/C++ code
9. **Upgrade to latest AVS SDK version.**
   1. Currently using **AVS SDK 1.11.0**, (12-19-2018). As of today, the latest version is **AVS SDK 1.12.0** (02-28-2019).

**References**

Below are links used for this FT900 Alexa Demo:

1. Alexa Voice Service (AVS)

<https://developer.amazon.com/alexa-voice-service>

1. Getting Started with the AVS Device SDK <https://www.youtube.com/watch?v=F5DixCPJYo8&feature=youtu.be>
2. Amazon’s Official AVS Device SDK (written in C++)

<https://github.com/alexa/avs-device-sdk>

1. AVS SDK Installation Guide on Raspberry PI (RPI)

<https://github.com/alexa/avs-device-sdk/wiki/Raspberry-Pi-Quick-Start-Guide-with-Script>

1. SOX Sound Exchange utility (used for MP3 audio decoding)

<http://sox.sourceforge.net/>

1. G711 Audio Companding algorithms (used for u-law compression/expanding)

<https://en.wikipedia.org/wiki/G.711>

The modified AVS SDK is located at

<https://github.com/richmondu/FT900/tree/master/Alexa/Amazon%20Alexa%20Gateway/avs_sdk>