TI Designs: TIDEP-0083
Voice Triggering and Processing With Cloud Connection to IBM Watson® Reference Design

Description

This signal processing-based reference design uses a multi-microphone input to a digital signal processor (DSP)—either C5517 or C5545—to acquire a high-quality voice signal, recognize a trigger word, and record a voice command to send to IBM Watson® cloud-based services. Watson™ returns a transcription of the speech (voice command) that was recorded locally, and cloud connectivity is handled by the CC3220SF Wi-Fi ®-based microcontroller (MCU). The C55x DSP algorithms are beamforming (BF), adaptive spectral noise reduction (ASNR), multi-source selection (MSS), and dynamic range compression (DRC).

Resources

TIDEP-0083	Design Folder
TIDA-01470 (LMB)	Design Folder
PCM1864	Product Folder
BOOST5545ULP	Tools Folder
TMDSEVM5517	Tools Folder
CC3220SF-LAUNCHXL	Tools Folder
SPRC133	Tools Folder
SIMPLELINK-CC3220-SDK	Tools Folder

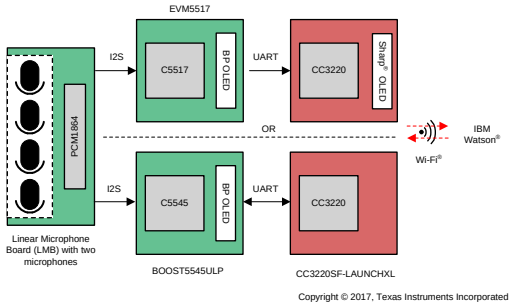
TI E2E™ Community — ASK Our E2E Experts

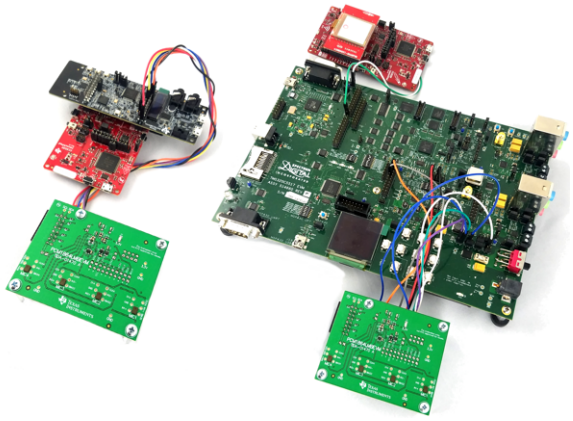
Features

- Multi-Microphone Support (Two or Four Microphones)
- Audio Pre-Processing Features
- IBM Watson Cloud Connection
- Sensory™ TrulyHandsFree™ Keyword Recognition
- Auto-Audio Recording After Keyword Triggering
- Speech-to-Text Conversion

Applications

- Voice-Activated Digital Assistant Products
- Voice-Activated Building Automation
- Video Doorbell
- Consumer Audio Products





An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 System Description

Voice communication is now a common way to interact with embedded devices. The Amazon Echo™ and Google Home™ create an upward-trending demand for voice-activated capability in consumer and industrial electronics. As a result, the market for voice-enabled embedded devices has increased significantly. Companies such as Amazon™, IBM®, Google®, and Microsoft® invest in their own ecosystems to enable more product development using voice as an interface; however, there are two fundamental design challenges with such embedded systems: poor quality of captured audio and limited on-chip resources to process audio.

This reference design discusses techniques from Texas Instruments address these challenges. Implementing these techniques ensures a quality voice-user interface experience along with an embedded system that is full of features.

Poorly-captured audio leads to an inferior user experience due to false triggers or unintelligible commands. A typical embedded system with a voice-user interface consists of a microphone array coupled with algorithms to process the captured audio commands. To address this challenge, signal processing techniques such as beamforming (BF) and noise reduction, can be used to improve system performance.

Another problem with voice-user interfaces is limited on-chip resources to simultaneously process audio and handle other functions in the system. Adding multi-language support or an expanded vocabulary consumes already limited on-chip resources. Leveraging cloud-based voice processing can overcome this limitation, which would allow for more feature integration in the device.

The key feature of the demonstrations highlighted in the TIDEP-0083 is the ability to perform real-time voice transcription. This functionality can be deployed in an embedded system that can make functional decisions based on the transcribed audio. An example of one such application would be a voice-activated thermostat where a user would say, "Set temperature 72°." The IBM Watson-transcribed audio can then be analyzed to signal the thermostat to set the temperature to 72°C.

1.1 Key System Specifications

Table 1. Key System Specifications

PARAMETER	SPECIFICATIONS	DETAILS
Linear microphone array	Two or four microphones out of the linear microphone board (LMB) can be used for the C5545 or C5517, respectively.	Section 2.2.4
PCM1864	PCM1864 audio ADC provides interfaces to the evaluation module(EVM). Each PCM1864 supports up to four audio microphones. Systems with more than four microphones require multiple PCM1864 devices.	Section 2.2.12
TMDSEVM5517 DSP EVM	Evaluation board (EVB) based on the C5517 DSP.	Section 2.2.1
C5545 BoosterPack™	EVB based on the C5545 DSP.	Section 2.2.2
Sharp® memory LCD BoosterPack (430BOOST-SHARP96)	Memory LCD BoosterPack plug-in module based on the LS013B4DN04 display from Sharp Electronics and features capacitive touch sliders.	Section 2.2.5
SimpleLink™ Wi-Fi CC3220SF wireless MCU LaunchPad™ development kit	EVB based on the CC3220SF single-chip, Wi-Fi wireless MCU.	Section 2.2.3
Chip support library (CSL)	Standard TI software release for the C55x family.	Section 2.2.6
Sensory TrulyHandsFree (THF)	Sensory TrulyHandsFree keyword recognition library	Section 2.2.7
SimpleLink Wi-Fi CC3220 Software Development Kit (SDK)	Standard TI software release for the CC3220 family.	Section 2.2.8
Executable c55xx_voiceuicloud	C5517 or C5545 DSP executable that processes multiple microphones streaming audio and generates a virtual-directional microphone audio stream. The processed audio is sent to the CC3220SF for transmission to the cloud.	Section 2.2.9
IBM_Voice_Recog_Demo_CC3220SF_LAUNCHXL_tirtos	CC3220SF executable that transmits the captured audio clip to the IBM Watson servers for voice transcription.	Section 3.1.2.2
Application source code and Code Composer Studio™ (CCS) projects	Source code for the data path unit test and for the applications that enables the user to modify or rebuild the code.	Section 2.2.10
TI audio libraries (or TELECOMLIB)	TI-optimized audio processing AEC-AER and VOLIB libraries.	Section 2.2.11
CCS version 7.2	TI-integrated development environment (IDE) that is used to run the executables and can be used to build the executables. The project was built and tested with CCS version 7.2 and code generation tools CGT for 5500 version 4.4.1 or higher. (It is assumed that the user is familiar with CCS).	Section 6
TI tools and utilities	A set of tools and utilities that can be downloaded from ti.com.	Section 2.2.13

2 System Overview

2.1 Block Diagram

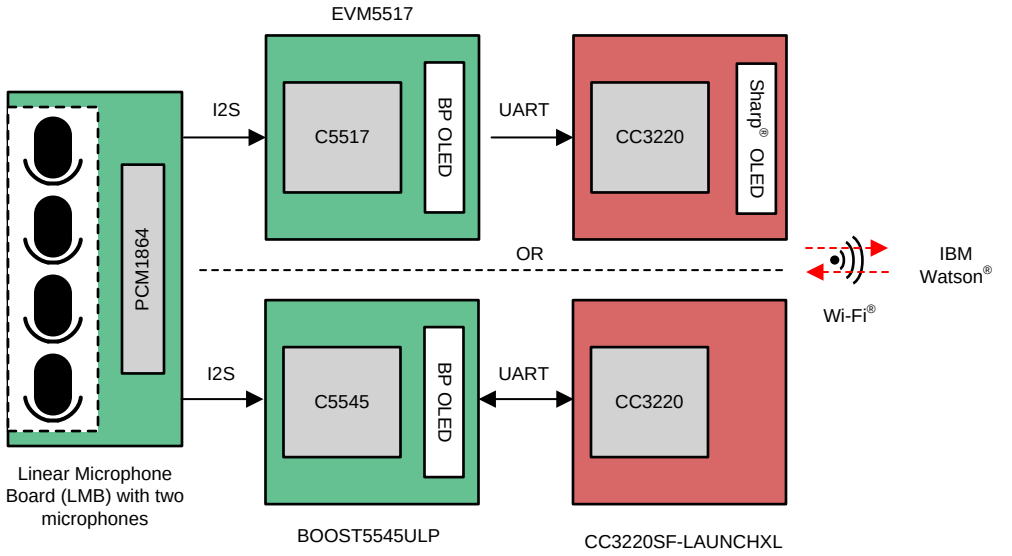


Figure 1. TIDEP-0083 Block Diagram

2.2 Highlighted Products

2.2.1 TMDSEVM5517

The TMDSEVM5517 EVM is based on the TMS320C5517 processor. For a full description of the TMDSEVM5517, see TMDSEVM5517. Figure 2 shows a block diagram of the TMS320C5517 DSP.

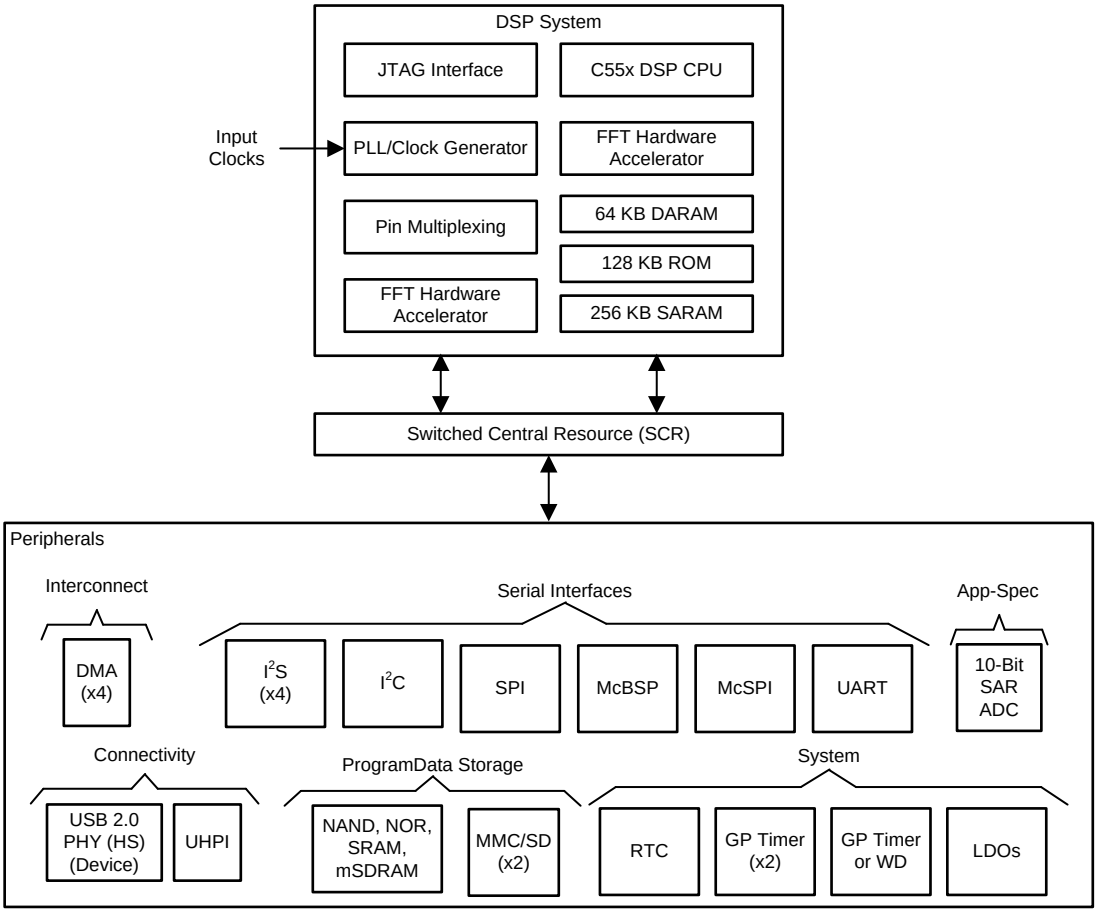


Figure 2. TMS320C5517 DSP Block Diagram

2.2.2 BOOST5545ULP

The BOOST5545ULP is an EVM based on the TMS320C5545 processor. For a full description of the BOOST5545ULP, see BOOST5545ULP. Figure 3 shows a block diagram of the TMS320C5545.

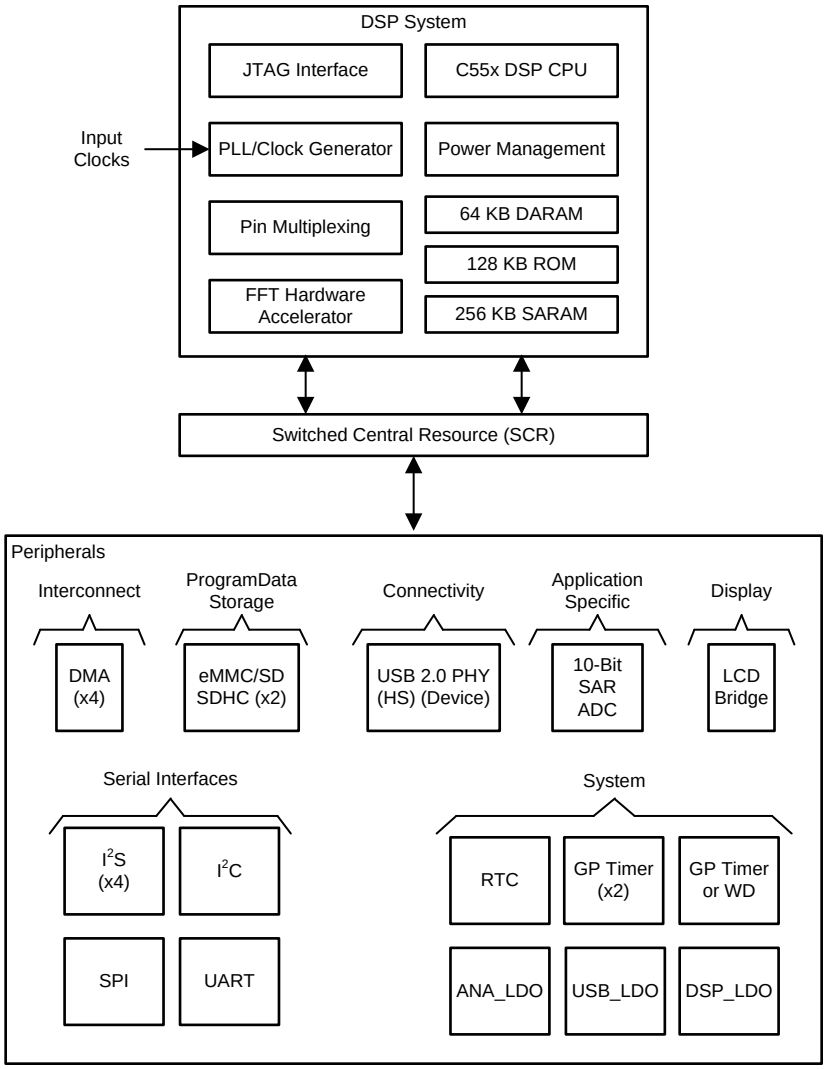


Figure 3. TMS320C5545 Block Diagram

2.2.3 CC3220

The CC3220 device is part of the SimpleLink MCU platform, which consists of Wi-Fi, *Bluetooth* ® low energy, Sub-1 GHz, and host MCUs that all share a common, easy-to-use development environment with a single-core SDK and rich tool set. A one-time integration of the SimpleLink platform allows the addition of any combination of the portfolio's devices into a design, which allows 100% code reuse when design requirements change. For more information, visit Overview for SimpleLink solutions.

Start the Internet-of-Things (IoT) design with a Wi-Fi CERTIFIED ™, single-chip, MCU System-on-Chip (SoC) with built-in Wi-Fi connectivity. Primarily created for IoT applications, the SimpleLink CC3220x device family from Texas Instruments is a single-chip solution that integrates two physically separated, on-chip MCUs.

The SoC architecture consists of an application processor— ARM® Cortex®-M4 MCU with a user-dedicated 256KB of RAM and an optional 1MB of XIP flash. The system network processor MCU runs all Wi-Fi and Internet logical layers. The ROM-based subsystem includes an 802.11b/g/n radio, baseband, and MAC with a powerful crypto engine for fast, secure Internet connections with 256-bit encryption.

The CC3220x wireless MCU family is part of the second generation of TI's Internet-on-a-chip™ family of solutions. This generation introduces new features and capabilities that further simplify the connectivity of things to the Internet. The new capabilities including the following:

- IPv6
- Enhanced Wi-Fi provisioning
- Enhanced power consumption
- Enhanced file system security (supported only by the CC3220S and CC3220SF devices)
- Wi-Fi AP connection with up to four stations
- More concurrently opened BSD sockets; up to 16 BSD sockets, of which 6 are secure
- HTTPS support
- RESTful API support
- Asymmetric keys crypto library

The CC3220x wireless MCU family supports the following modes: station, AP, and Wi-Fi Direct®. The device also supports WPA2 personal and enterprise security. This subsystem includes embedded TCP/IP and TLS/SSL stacks, HTTP server, and multiple Internet protocols. The device supports a variety of Wi-Fi provisioning methods including HTTP based on AP mode, SmartConfig™technology, and WPS2.0.

The power-management subsystem includes integrated DC-DC converters that support a wide range of supply voltages. This subsystem enables low-power consumption modes for extended battery life, such as low-power deep sleep, hibernate with RTC (consuming only 4.5 µA), and shutdown mode (consuming only 1 µA).

The device includes a wide variety of peripherals, such as a fast parallel camera interface, I2S, SD, UART, SPI, I²C, and four-channel ADC.

The SimpleLink CC3220x device family comes in three different device variants: CC3220R, CC3220S, and CC3220SF.

The CC3220R and CC3220S devices include 256KB of application-dedicated embedded RAM for code and data, ROM with external serial flash bootloader, and peripheral drivers.

The CC3220SF device includes application-dedicated 1MB of XIP flash and 256KB of RAM for code and data, ROM with external serial flash bootloader, and peripheral drivers. The CC3220S and CC3220SF device options have additional security features, such as encrypted and authenticated file systems, user IP encryption and authentication, secured boot (authentication and integrity validation of the application image at flash and boot time), and more.

The CC3220x device family is a complete platform solution including software, sample applications, tools, user and programming guides, reference designs, and the E2E™ online community. The device family is also part of the SimpleLink MCU portfolio and supports the SimpleLink developers ecosystem.

2.2.4 Linear Microphone Board (LMB)

Four microphones are mounted equidistant at a linear geometry on the microphone board. The PCM1864 samples the microphones and streams the digital values using I2S interfaces to the TMDSEVM5517 or BOOST5545ULP. Building a generic microphone array and calculating the filter coefficients associated with the linear array is described later in Section 3.1.2.8.1

2.2.5 Sharp Memory LCD BoosterPack™

The Sharp memory LCD BoosterPack plug-in module is based on the LS013B4DN04 display from Sharp Electronics and features capacitive touch sliders. The LCD displays the transcribed text from IBM Watson in the C5517 and CC3220SF setup.

2.2.6 CSL C55xx_csl

The CSL library contains utilities and drivers that are used to configure control and use all the peripherals and IP that are part of the C5517 and C5545 chip as well as the TMDSEVM5517 and BOOST5545ULP peripherals. Free download of CSL is available at *C55x Chip Support Libraries (CSL) Download* [8]. The version of C55xx_csl should be 3.08.00 or higher.

2.2.7 Sensory™ TrulyHandsFree™ Library

Sensory's TrulyHandsFree voice control technology builds upon the initial success of the TrulyHandsFree trigger and now offers multiple phrase technology that recognizes, analyzes, and responds to dozens of keywords. The library provided by TI is a 30-day evaluation version tied to the keyword phrase *Hello Blue Genie*. The library can be downloaded at *Sensory TrulyHandsFree Library download*[9].

2.2.8 SimpleLink™ Wi-Fi CC3220 SDK

The SimpleLink Wi-Fi CC3220 SDK contains drivers for the CC3220 programmable MCU and documentation required to use the solution. It also contains the flash programmer, a command line tool for flashing software, configuring network and software parameters (SSID, access point channel, network profile, and so on), system files, and user files (certificates, web pages, and so on). This SDK can be used with TI's SimpleLink Wi-Fi CC3220 LaunchPads.

Features:

Internet-on-a-chip sample applications:

- Email from SimpleLink Wi-Fi
- Information center: Get time and weather from the Internet
- https server: Host a secure web page on SimpleLink Wi-Fi
- XMPP: IM chat client
- Serial interface

Wi-Fi sample applications:

- Easy Wi-Fi configuration
- Station, AP modes
- TCP/UDP
- Security—Enterprise and personal, TLS/SSL
- Power management—Deep sleep, hibernate

MCU peripheral sample applications:

- Including parallel camera, I2S audio, ADC, I²C, PWMs, JTAG Flashing, and more.

2.2.9 c55xx_voiceuicloud Project

The c55xx_voiceuicloud project is part of the C55xx_csl release. The project gets two or four audio streams from the microphones array and applies beamforming, ASNR, and MSS to obtain a single, virtual-directional microphone directed at speech and to clean out clutter from a noisy environment. Using the C5517's onboard headphone jack, the demonstration is set up so the processed audio is output through the left channel of the stereo output and the unprocessed audio on the right channel. This setup enables the user to analyze the processed and unprocessed audio independently. There is no audio output on the headphone jack for the C5545-based demonstration.

2.2.10 Application Source Code and CCS Projects

The C55xx_csl release contains a CCS project and all the source code that is required to build the c55xx_voiceuiclouds project. Instructions how to build the project are given in Section 3.2.

2.2.11 TI Audio Libraries

TI audio libraries (TELECOMLIB) consists of two optimized libraries that are used in this reference design: the Acoustic Echo Cancellation-Removal (AEC-AER) library and the Voice Library (VOLIB). In addition, there is a DSPLIB package available for C55x devices, which contains many signal processing optimized algorithms. AEC-AER and VOLIB can be downloaded from ti.com's TELECOM tools folder. The user must install the audio libraries in the same directory as the c55_csl_3.08 installation. The library versions used are the following:

- AER LIB for C55X CPU version 3.3, version 17.00.00.00, or higher
- The VOLIB for C55X CPU version 3.3, version 2.01.00.01, or higher

2.2.12 PCM1864

The PCM1864 is a 103-dB, two stereo channel (four channels total), software-controlled audio ADC with universal front end. For a full description of this device, see the PCM1864 product page.

2.2.13 Set of Tools and Utilities

Table 2 lists the set of TI tools and utilities that are required for building the c55xx_voiceuicloud project.

Table 2. Required Software Dependencies (1) (2)

TOOLS AND UTILITY NAME	LOAD LOCATION
DSP BIOS version 5.42.2.10	DSPBIOS 5.42.2.10
XDAIS version 7.24.0.4	XDAIS 7.24.0.4
XDC TOOLS version 3.24.05.4	XDC TOOLS
Uniflash Tool v4.1 or later	UniFlash Tool
Sensory THF 1.0.0(3)	Sensory THF Library

(1) The user must install AEC-AER and VOLIB libraries as well as the tools from Table 2 in the same directory that C55xx_csl was installed.

(2) For a complete set of version requirements, see the list of dependencies in the release notes for C55X_CSL.

(3) The Sensory THF library installs at *C:\ti\c55_lp\sensory* by default. Once Sensory THF has been installed, copy the sensory directory and place it in *C:\ti\c55_lp\c55_csl_3.08\demos\voice_ui_cloud\third_party*. This step is required to ensure a successful build of the voice_ui_cloud CCS projects.

2.3 Design Considerations

2.3.1 More About Beamforming (BF)

This reference design uses a BF algorithm to form a virtual-directional microphone that points to the direction of the speaker or the desired audio source. The BF algorithm amplifies the speech signal from the desired direction, and attenuates all signals from all other directions. In addition to BF, TI offers a set of audio algorithms that may further improve the quality of sound-like dynamic range compression. An overview of the audio BF mathematics and algorithm can be found in *Acoustic Source Localization and Beamforming: Theory and Practice* [1] and *Beamforming* [2] on Wikipedia. A group of microphones are mounted at predefined locations, which are either along a straight line or on a circle. A point sound source reaches different microphones with different phase delays. The phase delay depends on the frequency, the speed of sound, the distance between each microphone, and the sound source. The distances between the source and the microphones are function of the direction. Figure 4 shows the distance and the phase difference between two microphones as a function of the direction of the signal arrival.

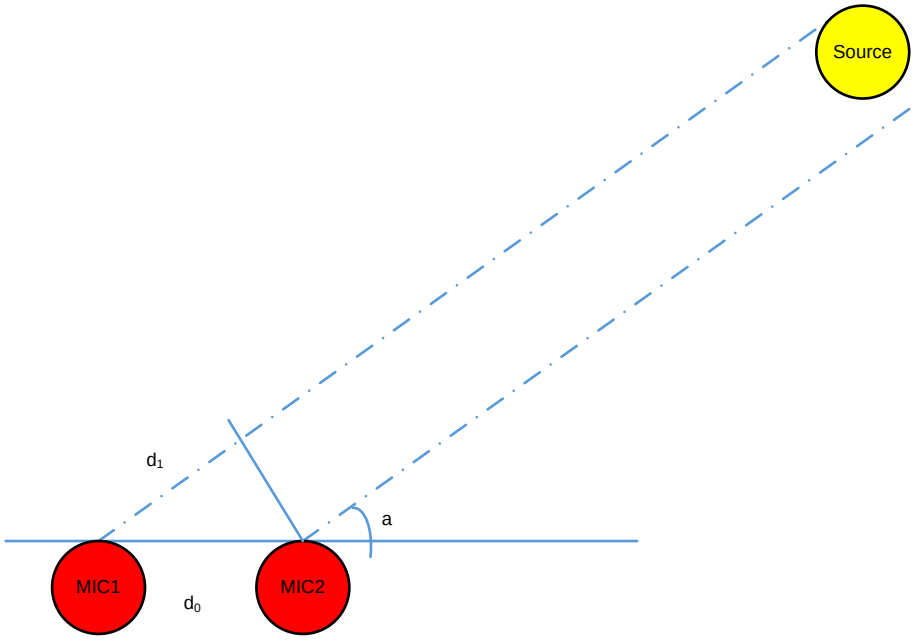


Figure 4. Differences in Distance, Time, and Phase Between Two Microphones

From Figure 4, d_1 is calculated in Equation 1.

$$d_1 = d_0 \times \cos(\alpha) \tag{1}$$

The signal time difference, Δ_t, between mic1 and mic2 is d_1 divided by the speed of sound, as shown in Equation 2.

$$\Delta_t = \frac{d_1}{sos} \tag{2}$$

The phase difference between mic1 and mic2 is shown in Equation 3.

$$\Delta_0 = 2 \times \pi \times \Delta_t \times f = 2 \times \pi \times f \times \frac{d_1}{sos} = 2 \times \pi \times f \times d_0 \times \frac{\cos(\alpha)}{sos} \tag{3}$$

Where:

- Δ_θ is the phase difference
- f is the signal frequency
- d_0 is the distance between two microphones
- α is the angle of arrival and sos is the speed of sound

In a multi-microphone BF system, the algorithm applies a set of delay filters to the microphones' signals to shift the signal phase and get the same phase for all the signals (from all microphones) that arrive from one direction. The contribution of all filtered microphones signals are summed together. Thus, the process amplifies signals that arrive from that direction. Because the phase shift (see Equation 3) depends on the angle of arrival (AOA), the phases of filtered signals that arrive from other directions are not the same. Therefore, the sum of all the signals from another direction is decreased, and the energy of the noise (undesired signal that comes from another direction) is reduced. From Equation 3, it is clear that the quality of the reduction of noise depends on the noise frequency. While the BF filters are designed to reduce noise from typical mid-range and higher frequencies, low-frequency noise are not reduced. An adaptive ASNR filter is applied to reduce the effect of low-frequency noise.

Figure 5 shows the reduction of noise as a function of frequency when the BF and the ASNR are applied.

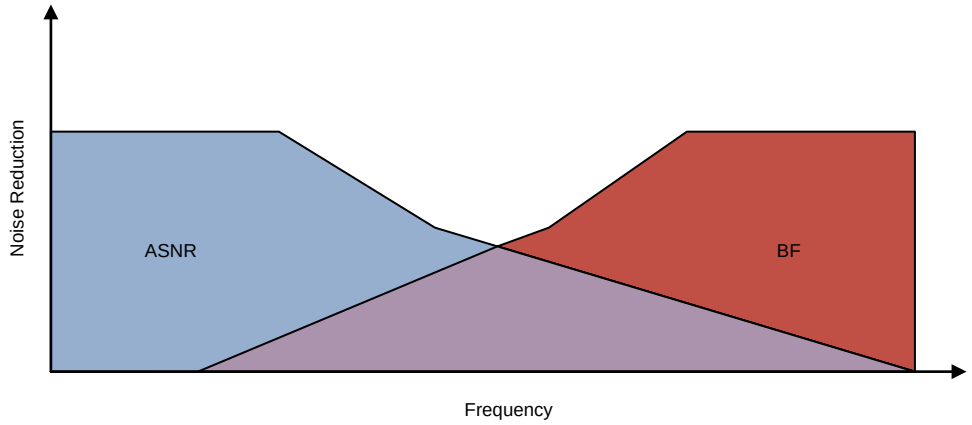


Figure 5. ASNR and BF Noise Reduction as Function of Noise Frequency

2.3.1.1 Multi-Angle Beamforming (BF)

Figure 6 shows a typical multi-angle BF where multiple BF delay filters and ASNR filters are applied to the microphone sets' data. Each BF and ASNR output corresponds to a different AOA and behaves like a directional microphone; therefore, it is called virtual microphone. An MSS algorithm chooses the best fit virtual microphone.

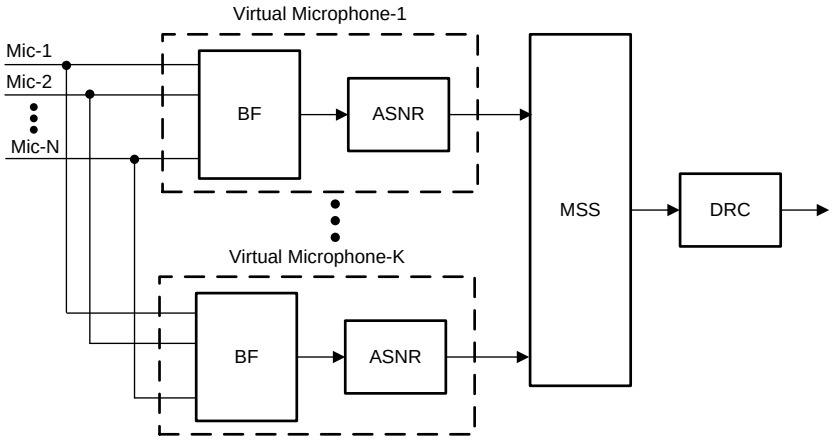


Figure 6. Multi-Angle BF System

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Required Hardware and Software

To provide options in a system implementation, TIDEP-0083 provides two C55x demonstrations with the CC3220SF LaunchPad:

1. C5517-based demonstration with four microphone inputs:
   - Four microphone inputs
   - Sensory *Hello Blue Genie* trigger word integrated
   - No return UART path to C5517
   - Transcribed text shown on Sharpmemory LCD BoosterPack attached to the CC3220SF LaunchPad
2. C5545-based demonstration with two microphone inputs:
   - Two microphones input
   - Sensory *Hello Blue Genie* trigger word integrated
   - UART return path to C5545
   - Transcribed text shown on BOOT5545ULP OLED
   - Compact hardware setup with BOOST5545ULP and CC3220SF LaunchPad stacked on top of each other

NOTE: Each demonstration runs independently and allows the user to evaluate the C5517 and C5545 with the IBM Watson voice transcription service.

3.1.1 Hardware

3.1.1.1 EVM5517 + CC3220SF-LAUNCHXL

This section provides instructions on running the voice transcription demonstration with the EVM5517,CC3220SF-LAUNCHXL with the transcribed text displayed on the Sharp memory LCD BoosterPack connected to the CC3220SF-LAUNCHXL. Figure 7 shows an overview of the demonstration.

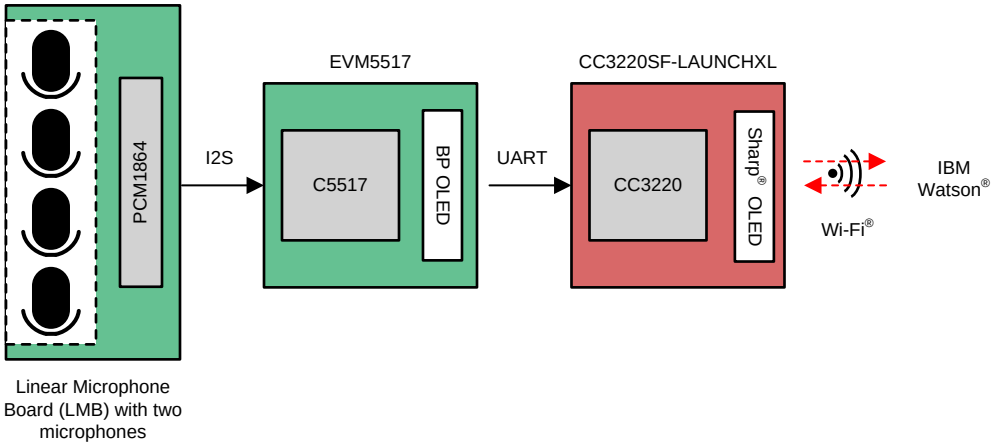


Figure 7. C5517-Based Voice Transcription Demonstration Block Diagram

3.1.1.1.1 Connection Details for Linear Microphone Array, TMDSEVM5517, CC3220SF-LAUNCHXL, and 430BOOST-SHARP96

This section covers the connection details for the hardware device in this demonstration.

Table 3 describes the signals that linear microphone array connects to the TMDSEVM5517. For a video demonstration of the hardware connections, see *Demonstrating Triggering and Control with Cloud Connection to IBM Watson*[10].

Table 3. LMB Microphone Signals

SIGNAL NAME	TMDSEVM5517	LMB PIN
3.3 V	J10_Pin9	3.3 V
Ground	J10_Pin5	GND
I2C SCL	J14_Pin16	LMB_SCL
I2C SDA	J14_Pin20	LMB_SDA
Bit clock (microphone one and microphone two)	J27_Pin3 (jumper off)	LMB_BCLK
Frame clock (microphone one and microphone two)	J27_Pin4 (jumper off)	LMB_LRCLK
Data one	J30_Pin2 (jumper off)	LMB_DATA
	J29_Pin1_Pin3 (jumper on)	
	J29_Pin2_Pin4 (jumper on)	
	J30_Pin1_Pin3 (jumper off)	
Bit clock (microphone three and microphone four)	J27_Pin1	LMB_BCLK
Frame clock (microphone three and microphone four)	J27_Pin2	LMB_LRCLK
Data three	J28_Pin2	LMB_DATA3
	J28_Pin1_Pin3 (jumper on)	
	J28_Pin2_Pin4 (jumper off)	
	UART_EN (jumper off)	
Master Mode	—	LMB_J3: Jumper ON
MIC BIAS	—	LMB_J10: Jumper ON
DIP Switch SW4	1-ON 2-OFF 3-OFF 4-OFF	—
DIP Switch SW3	1-OFF, 2-ON, 3-OFF, 4-ON, 5-OFF, 6-OFF, 7-OFF, 8-ON	—
DIP Switch SW5	1-ON 2-ON 3-ON 4-ON	—
DIP Switch SW6	1-ON 2-ON 3-ON 4-ON	—
DIP Switch SW11	1-OFF, 2-OFF, 3-OFF, 4-OFF, 5-ON, 6-ON	—

Figure 8 shows the wiring of the EVM5517 and LMB.

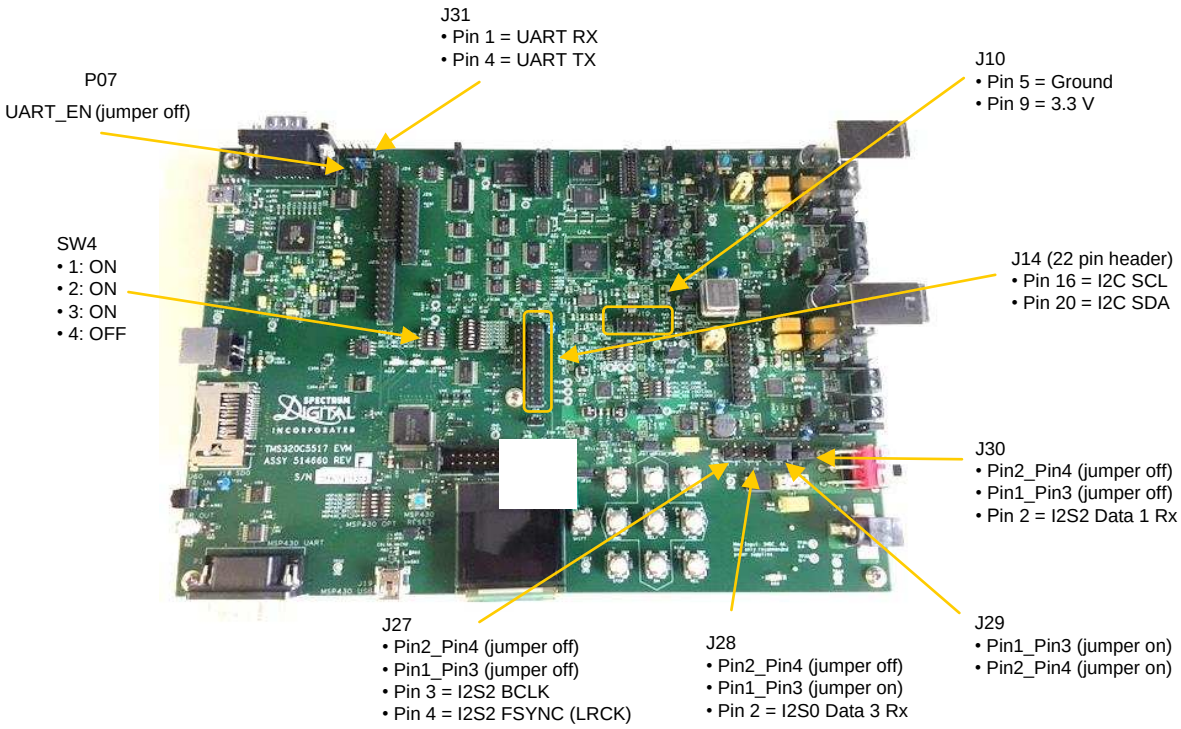


Figure 8. EVM5517 and LMB Wiring Illustration

Table 4 describes the signals that connects the TMDSEVM5517 to CC3220SF-LAUNCHXL.

Table 4. Connection Details TMDSEVM5517 to CC3220SF-LAUNCHXL

SIGNAL NAME	TMDSEVM5517	CC3220SF-LAUNCHXL
UART TX	J31 Pin4	P1-P02
UART RX	J31 Pin1	P1-P01
Ground	J24-Pin21	P2-GND

Figure 9 shows how to connect the CC3220SF-LAUNCHXL to the 430BOOST-SHARP96 LCD.

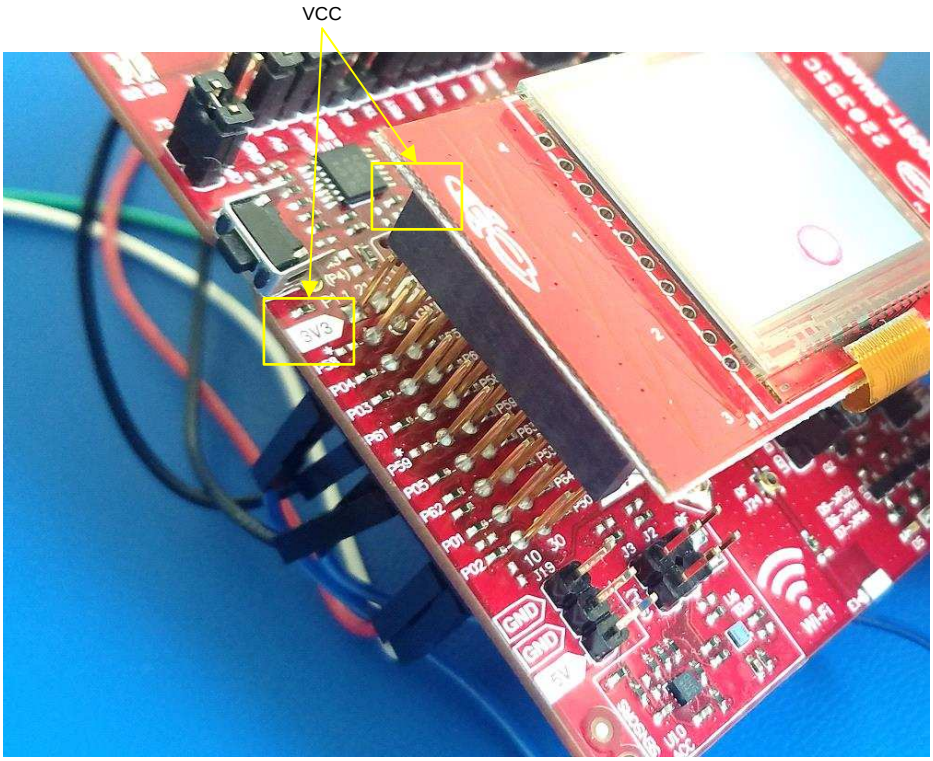


Figure 9. Connecting CC3220SF-LAUNCHXL to 430BOOST-SHARP96

Table 5 describes the CC3220SF-LAUNCHXL jumper connections for the Sharp LCD.

Table 5. CC3220SF-LAUNCHXL Jumper Connections for Sharp® LCD

CC3220SF-LAUNCHXL	CC3220SF-LAUNCHXL
SPI CS (P03)	LCD CS (P59 NC)
GPIO 6 (P58)	LCD En (P58 NC)

Figure 10 shows the jumper wires required on the CC220SF-LAUNCHXL.

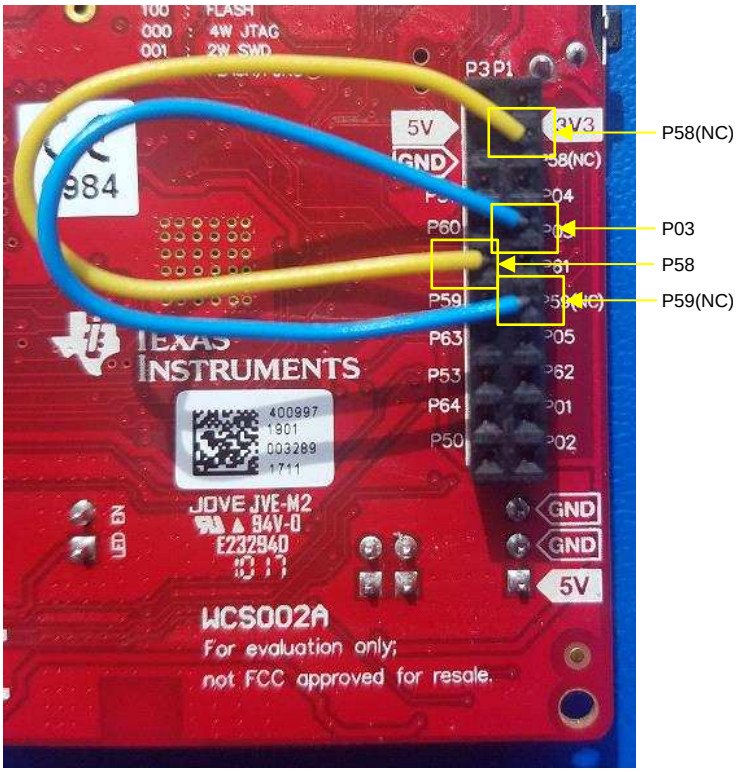


Figure 10. Jumper Wires Required on CC3220SF-LAUNCHXL

3.1.1.2 BOOSTC5545ULP + CC3220SF-LAUNCHXL

This section provides instructions on running the voice transcription demonstration with the BOOST5545ULP and CC3220SF-LAUNCHXL with the transcribed text displayed on the BOOST5545ULP OLED. Figure 11 shows an overview of the demonstration.

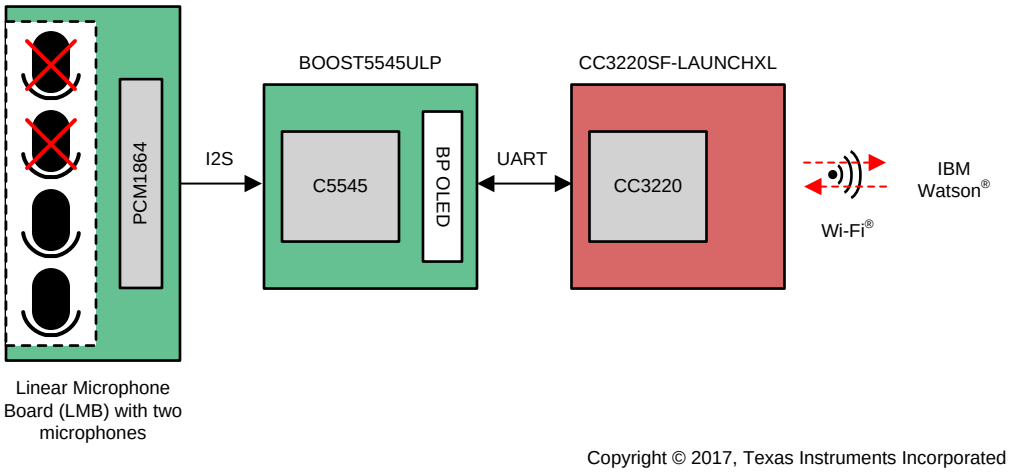


Figure 11. C5545-Based Voice Transcription Demonstration Block Diagram

3.1.1.2.1 Connection Details for Linear Microphone Array, BoostC5545ULP, and CC3220SF-LAUNCHXL

Below are the connection details for each piece of hardware in this demonstration.

Table 6 describes the signals that linear microphone array connects to the BOOST5545ULP. For a video demonstration of the hardware connections, see *Demonstrating Triggering and Control with Cloud Connection to IBM Watson*[10].

Table 6. LMB Microphone Signals

SIGNAL NAME	BOOST5545ULP	LMB PIN
3.3 V	J7_Pin1	3.3V
Ground	J7_Pin4	GND
I2C SCL	J7_Pin17	LMB_SCL
I2C SDA	J7_Pin19	LMB_SDA
Bit clock	J7_Pin16	LMB_BCLK
Frame clock	J7_Pin14	LMB_LRCLK
Data one	J7_Pin18	LMB_DATA

Table 7 and Table 8 describe the jumper details for CC3220SF-LAUNCHXL and BOOST5545ULP.

Table 7. CC3220SF-LAUNCHXL Jumper Details

CC3220SF-LAUNCHXL	JUMPER
Onboard accelerometer	J2 - OFF
Onboard temperature sensor	J3 - OFF
LED EN	J9 - ON

Table 8. BOOST5545ULP Jumper Details

BOOST5545ULP	JUMPER
UART RX	JP2 - jumper ON pins 3 and 4
UART TX	JP3 - jumper ON pins 1 and 2

Figure 12 illustrates how to connect the additional jumpers under the CC3220SF-LAUNCHXL.

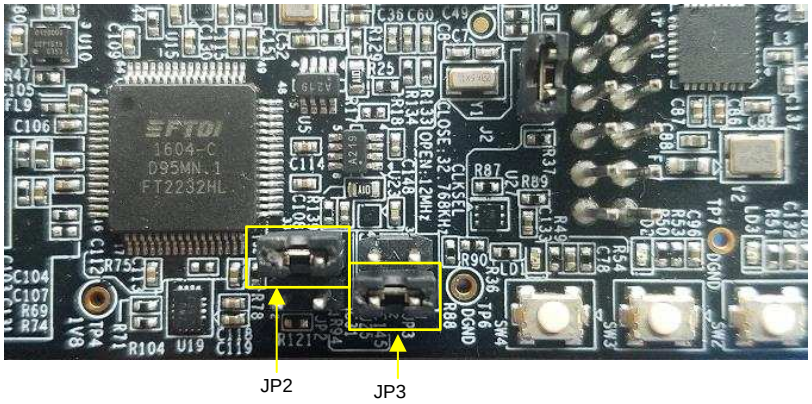


Figure 12. BOOST5545ULP UART Jumper Details

Figure 13 and Figure 14 show how the two boards would be stacked on top of each other. Once stacked, the LaunchPad headers facilitate the required connections between the two boards.

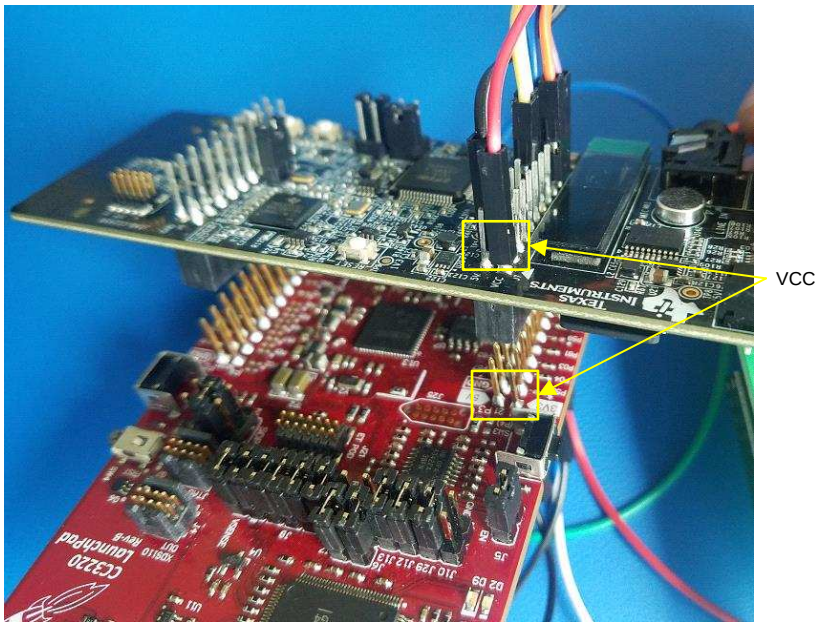


Figure 13. Connecting CC3220SF-LAUNCHXL to BOOST5545ULP

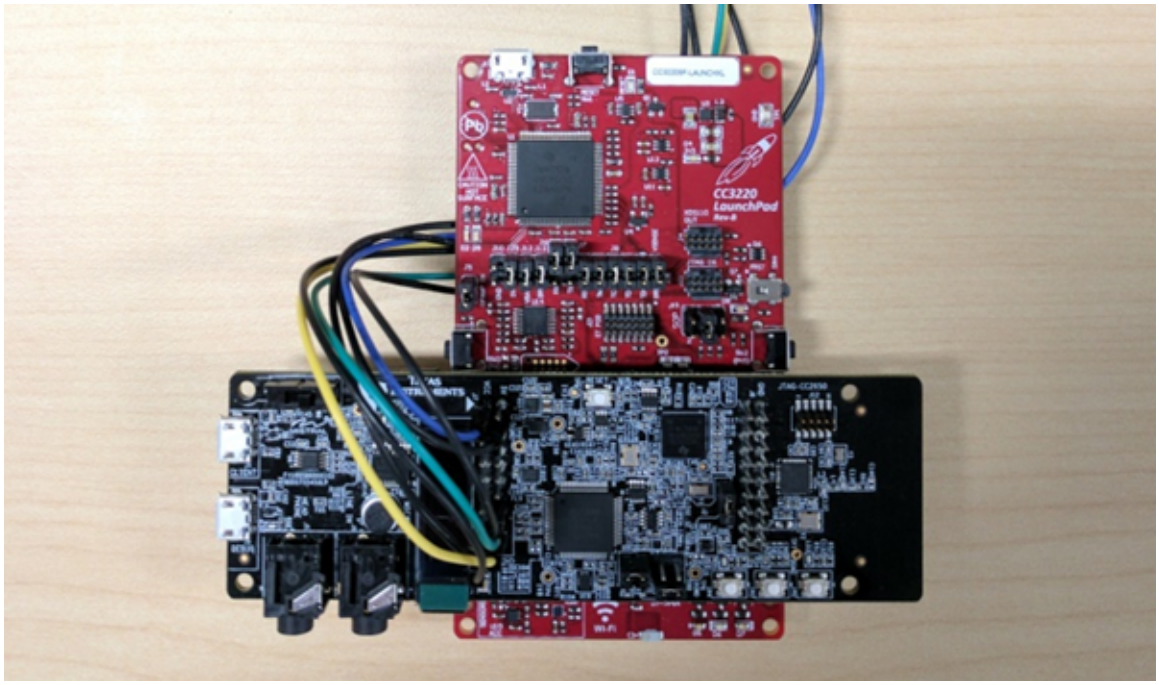


Figure 14. Connecting CC3220SF-LAUNCHXL to BOOST5545ULP (Stacked)

Table 9 describes the CC3220SF-LAUNCHXL jumper connections for the BOOST5545ULP UART.

Table 9. CC3220SF-LAUNCHXL Jumper Connections for BOOST5545ULP UART

CC3220SF-LAUNCHXL	CC3220SF-LAUNCHXL
P08	P04
P07	P03

Figure 15 show the jumper wires required on the CC3220SF-LAUNCHXL.

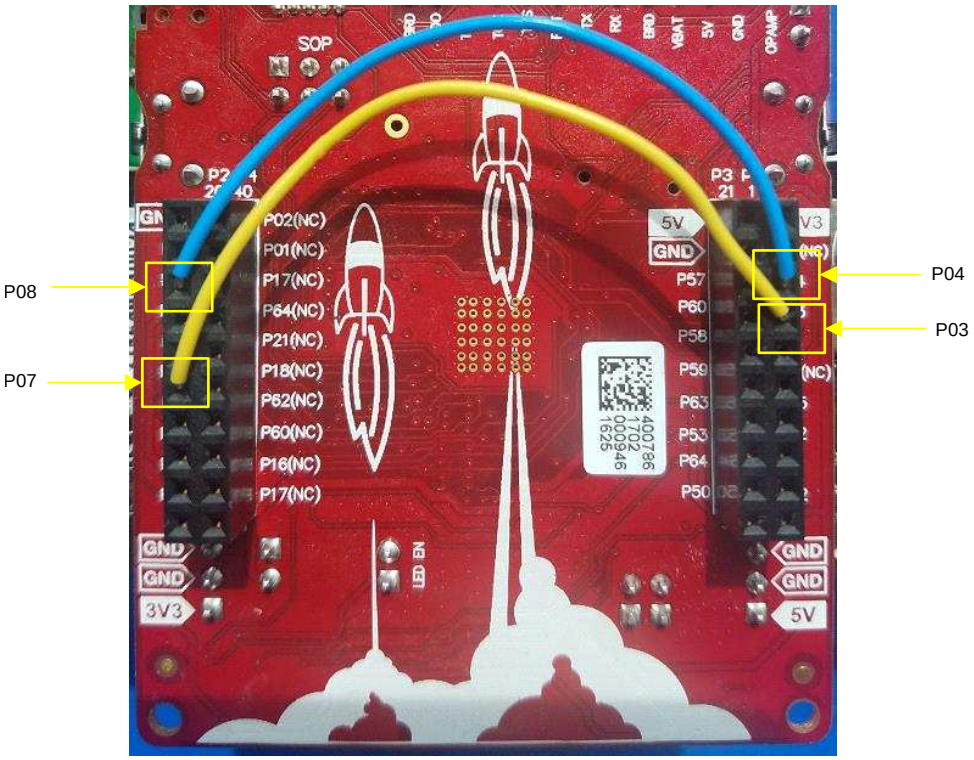


Figure 15. Jumper Wires Required on CC3220SF-LAUNCHXL

3.1.2 Software

3.1.2.1 C55X Application Source Code and CCS Projects

The TIDEP-0083 design consists of two C55x demonstrations as described by Figure 1. Both demonstrations capture audio from the LMB and transmits to IBM Watson for transcription through the CC3220SF LAUNCHXL. The C55x device must be programmed with the application binary in order for the system to work.

3.1.2.1.1 C5517 Version of Demonstration

- Uses a Sharp LCD on the CC3220SF-LAUNCHXL displaying the transcribed text
- Uses four microphones on the LMB
- *Hello Blue Genie* (sensory) voice trigger to begin recording
- Has more MIPS (200MHz) available to accommodate additional algorithms.
- Has a demonstration available as a CCS project or stand-alone binary distributed as part of the C55x CSL <CSL INSTALLATION LOCATION>\demos\voice_ui_cloud\c5517

See Section 3.2 for details on importing and building the demonstration in CCS.

3.1.2.1.2 C5545 Version of Demonstration

- Uses the C5545 BoosterPack OLED to display transcribed text
- Uses two microphones on the LMB
- *Hello Blue Genie* (sensory) voice trigger to begin recording
- Has a small form factor as the BOOST5545ULP and CC3220SF-LAUNCHXL are stacked
- Has a demonstration available as a CCS project or stand-alone binary distributed as part of the C55x CSL <CSL INSTALLATION LOCATION>\demos\voice_ui_cloud\c5545

See Section 3.2 for details on importing and building the demonstration in CCS.

3.1.2.1.3 C55x DSP BIOS Task Summary

The demonstration is DSP BIOS based with several tasks. The following is a high-level description of each task in the C5545 program:

- audioTsk—Task that collects the audio from the microphones through I2S DMA and provides it for audio processing (BF, and so on) and voice-triggering engine.
- oledTsk—Task that sends the string received from UART and displays it on the OLED.
- TSK_blinkXGvoicerecog—Task that prints the trigger phrase on the OLED.
- TSK_Recognizer—Sensory keyword recognition task; this task takes audio frames and feeds it into the Sensory keyword engine and awaits a positive or negative result.
- uartdmaTsk—Task that outputs sends the audio through UART to the CC3220SF-LAUNCHXL.
- TSK_blinkXFvoicerecog—This task blinks the XF LED on the EVM when there is a positive keyword recognition (applies only to the C5517 demonstration).

3.1.2.2 CC3220 Application Source Code and CCS Projects

For TIDEP-0083, the CC3220 must be programmed with the application binary for the system to work. A demonstrationis available as a CCS project or stand-alone binary distributed as part of the C55x CSL <CSL INSTALLATION LOCATION>\demos\voice_ui_cloud\cc3220.

The demonstration application provided is built with TI-RTOS as its real-time operating system (RTOS), which allows for discrete tasks to be handled and scheduled concurrently.

In this application, there are four tasks that are spawned that exist concurrently but are run using the preemptive scheduler built into TI-RTOS, which allows for real-time handling of tasks according to priority.

The following is a high-level description of each task in the CC3220 program:

- Main

  This task is the task that is initially started after reset. This thread spawns the mainLoopTask and then starts the scheduler.
- mainLoopTask

  This task is the where the core processing loop of the application is handled. This task sets up and handles all of the audio input from the C55X in addition to having the core logic.
- provisioningTask

  This task handles the Wi-Fi provisioning state machine. In the event where the CC3220 is not provisioned to an available Wi-Fi network, the provisioning task runs and interacts with the SimpleLink Starter Pro app to provision the C3220. While the device is unprovisioned, the mainLoopTask and the wifiTask are blocked. Once the C3220 is provisioned, this thread posts a semaphore to the other tasks indicating that provisioning is complete.

  If provisioning is disabled through the configuration option in the appConfig.h file, then the provisioningTask is not spawned and the CC3220 joins the network given by the Wi-Fi parameters defined in appConfig.h at compile time.
- wifiTask

  This task handles all of the SimpleLink Wi-Fi calls and connects to the IBM Watson service to send voice data collected by the mainLoopTask and receive back a text transcription of the audio. The wifiTask makes use of an HTTP library to handle the low-level SimpleLink API calls, but implements

the HTTP POST logic.

The demonstration application is located in the <CSL INSTALL PATH>\c55_csl_x.xx\demos\voice_ui_cloud\cc3220 directory of the C55X CSL. Within that directory is the source of the project in the folder IBM_Voice_UI_Demo_CC3220SF_LAUNCHXL. In addition to the source, there is a .projectspec file that can be used to import the source project into CCS.

For ease of use, there are also pre-built application binaries located in \demos\voice_ui_cloud\cc3220. There are two separate versions of the binary: one for use with the C5517 and the other for the C5545. The two versions are similar, but have the following differences:

The C5517 version of the demonstration:

- Uses a Sharp LCD for output
- Must be connected to C5517 with jumper wires
- Has LEDs that illuminate to indicate debug information

The C5545 version of the demonstration:

- Uses the C5545 BoosterPack OLED for output
- Connects directly to the C5545 BoosterPack using the LaunchPad headers
- Has LEDs disabled due to pin conflicts

There are a few methods to load the CC3220 binary. This guide covers the following:

1. Programming the CC3220 with a preconfigured ImageCreator project using Uniflash:
   - This is the quickest and simplest method of loading the binary.
   - The Watson API key must be provided by programming an additional file to the CC3220.
   - The binary programmed is provided by default and cannot be customized without rebuilding from source.
   - The binary programmed uses a statically provisioned Wi-Fi connection with the following settings:
     – SSID = *simplelinktest*
     – Security key = *wifitest*
     – Security type = WPA2

   A Wi-Fi network with those settings must be used for proper functionality of the demonstration.
2. Programming the CC3220 by creating an ImageCreator project and then flashing with Uniflash:
   - This is a more reusable method than using a preconfigured image but does not require building the demonstration from source.
   - The Watson API key must be provided by programming an additional file to the CC3220.
   - The binary programmed is provided by default but can also accommodate binaries built from source.
   - The binary programmed by default following the instructions in Section 3.1.2.3.1 uses a statically provisioned Wi-Fi connection with the following settings:
     – SSID = *simplelinktest*
     – Security key = *wifitest*
     – Security type = WPA2

   A Wi-Fi network with those settings must be used for proper functionality of the demonstration.
3. Importing the source code of the CC3220 demonstration into CCS, building the image, and then loading the binary using the debugger

- This is a more involved method than using a prebuilt image, but allows for complete customization of the demonstration.
- The Watson API key must be provided by modifying the AUTHENTICATION define in the httpvars.h file as described by step 11 of Section 3.1.2.4.
- The binary built by default uses a statically provisioned Wi-Fi connection with the following settings:
  – SSID = *simplelinktest*

- Security key = *wifitest*
- Security type = WPA2

However, these Wi-Fi parameters can be freely changed to suit the network environment by modifying the appropriate parameters in appConfig.h.

- A full description of all of the customization options is available in Section 3.1.2.7 and Section 3.1.2.8.

Provided within the cc3220 directory is the following:

- CC3220SF_Uniflash_Project_C5517.zip and CC3220SF_Uniflash_Project_C5545.zip Uniflash project images that can be loaded directly into UniFlash and then programmed by following Section 3.1.2.3.1.
- A set of prebuilt binaries in \IBM_Voice_UI_Demo_prebuilt_binaries\ that can be loaded into Uniflash and then programmed by following Section 3.1.2.3.2.
- The source code for the CC3220 demonstration application in \IBM_Voice_UI_Demo_CC3220SF_LAUNCHXL\, which can be built and loaded in debug mode by following Section 3.1.2.2.1.

NOTE: An IBM Watson API key must be provided to the application regardless of the method used to load the binary.

To load the binary:

1. Provide an external file with the API key that programmed with UniFlash.
   - This is the method that must be used if using the prebuilt binaries. The API key must be provided as a plaintext ASCII file containing only the API key. This file must have filename “ibm_watson_key.txt”. Seeibm_watson_key_example.txt for an example of how to provide the API key.
2. Provide the API key in httpvars.h and then rebuilding the application.
   - This is the method that is enabled by default in the source code of the application. Before building, the API key must be provided by editing the AUTHENTICATION define in the httpvars.h file.

In either case, an IBM Watson API key must be first acquired by following the steps in Section 3.1.2.4. The CC3220 demonstration application will not function without supplying the IBM Watson API key.

3.1.2.2.1 Importing and Building CC3220 Project in CCS

The most flexible method to use the application is to build from the demonstration source code. This method allows editing of Wi-Fi parameters, easy changes to the IBM Watson key, and modification to the configuration settings of the demonstration.

Table 10 lists the set of TI tools and utilities that are required for building the CC3220 demonstration application:

Table 10. CC3220 Toolchain Prerequisites

TOOLS AND UTILITY NAME	LOAD LOCATION
CCS version 7.2 or later	CCS_7
CC3220 SDK version 1.40.00.03 or later	CC3220_SDK
Uniflash Tool v4.1 or later	Uniflash Tool

It is assumed that an IBM Watson API key is available. If an API key is not available, first complete Section 3.1.2.4.

1. Open CCS.
2. Import the provided .projectspec file by clicking on *Project→ Import CCS Project*. Then, click the *Browse* button in the window that pops up, navigate to <CSL_INSTALL_PATH>\c55_csl_x.xx\demos\voice_ui_cloud\cc3220\, and then press *OK*. There should be only one discovered project. Then click on *Finish* to import the project into the workspace.

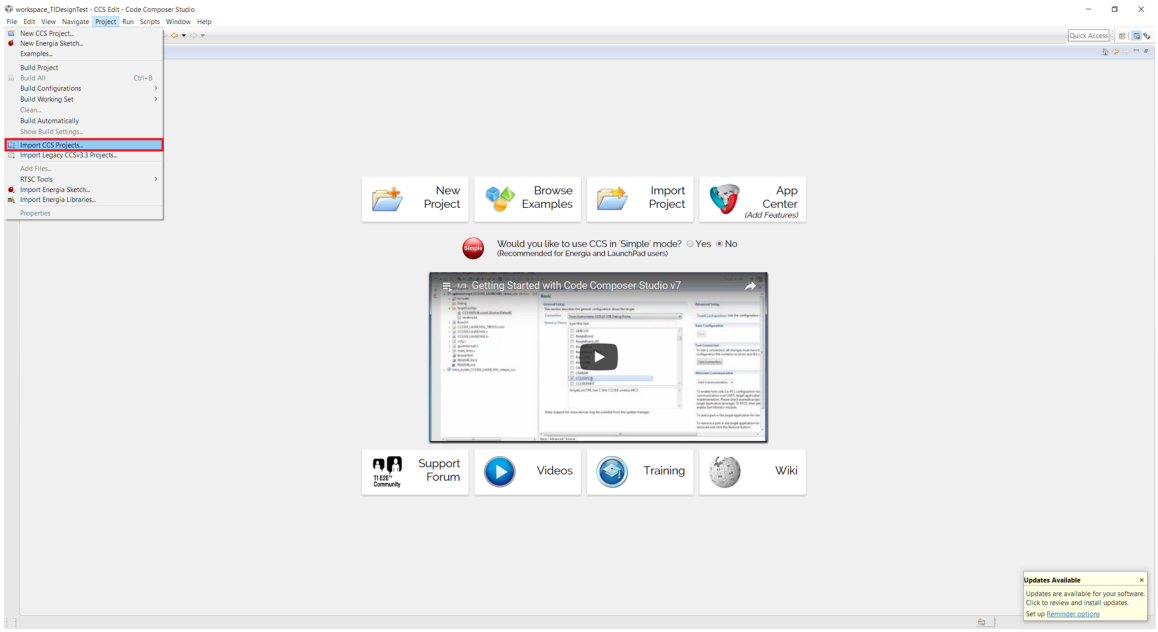


Figure 16. Import CCS Project

3. Expand the IBM_Voice_UI_DEMO_CC3220SF_LAUNCHXL project in the workspace and then open appConfig.h. This file is the main configuration file for the CC3220 demonstration with all of the non-HTTP configuration settings. There are a few parameters that need to be set manually, specifically:
   - SSID_NAME

     This parameter must be defined according to the network environment.
   - SECURITY_TYPE

     This parameter must be defined according to the network environment.
   - SECURITY_KEY

     This parameter must be defined according to the network environment.
   - UART_BAUDRATE

     This parameter must match the baud rate configured on the C55X device. By default, both the CC3220 and the C55X operate at 1,228,800 baud.
   - ENABLE_C5545_PIN_SETTINGS

     This parameter must be set to 1 if the demonstration system is using a C5545 BoosterPack and must be set to 0 if using a C5517 EVM.

See sections 3.1.2.7 and 3.1.2.8 for a full description of all provided configuration settings.

Once those defines are set, expand the src\ directory and open httpvars.h. The IBM Watson API key is provided in this file through the AUTHENTICATION define. Insert the API key, and then edit the DEVICE_YEAR/MONTH/DATE to reflect the current date.

1. In the Project Explorer pane, right-click on the *IBM_Voice_UI_Demo_CC3220SF_LAUNCHXL* project and then select *Rebuild Project*. The program must compile without any errors or warnings. If there are any problems with the build, ensure that the prerequisite tools are installed.

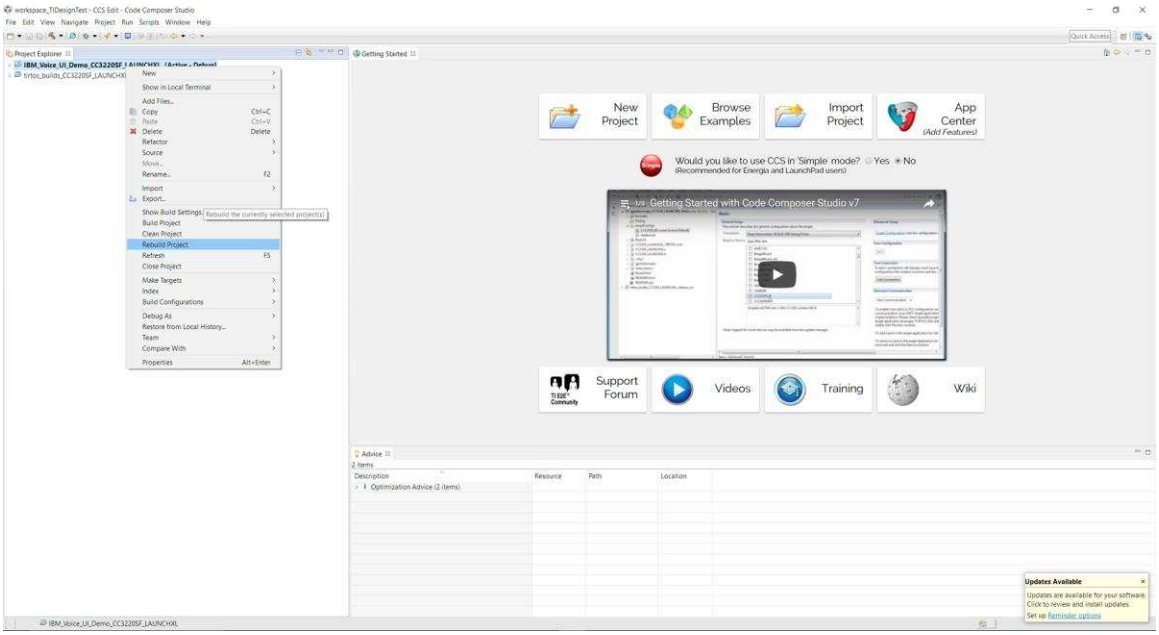


Figure 17. Rebuild CCS Project

2. Once the demonstration is built, the demonstration application can be run directly in CCS through the debugger. This is done by pressing on the Debug icon in the upper left of the CCS window.

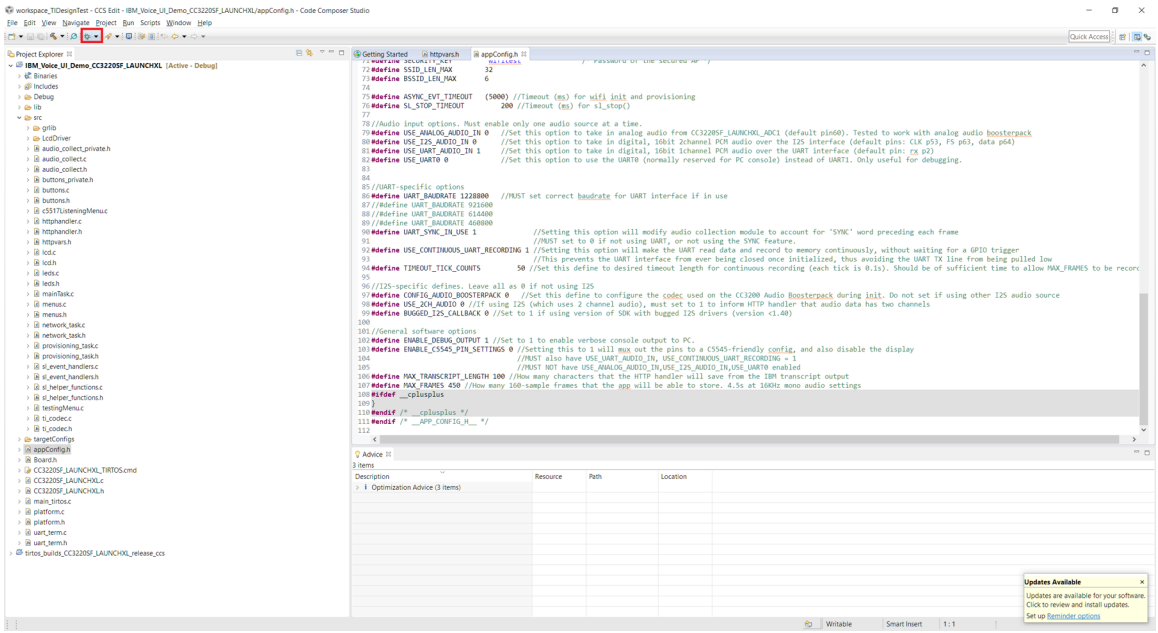


Figure 18. Starting CCS Debugger

Note that the demonstration relies on the geotrustglobalca.der certificate file being present on the file system of the device.Prior to running the program, follow the steps in Section 3.1.2.3.1 or Section 3.1.2.3.2 to load the CC3220 using UniFlash. After building the demonstration, one can also use the .bin file generated in <CCS Workspace Dir>\IBM_Voice_UI_Demo_CC3220SF_LAUNCHXL\Debug\ as the MCU image instead of the prebuilt binary file when following the Image Creator steps.

3.1.2.3 Flashing CC3220SF LaunchPad™ With Uniflash

This section describes two ways of programming the CC3220SF LaunchPad. Section 3.1.2.3.1 explains the process of programming the out-of-the-box demonstration by importing a preconfigured Image Creator project to UniFlash. Section 3.1.2.3.2 shows how to create an Image Creator project from scratch to program the CC3220SF with a binary generated from CCS.

The CC3220SF LaunchPad UniFlash images for the relevant demonstrations (C5517 or C5545) can be found in the C55x CSL package at <CSL INSTALL PATH>\c55_csl_x.xx\demos\voice_ui_cloud\cc3220. In Section 3.1.2.3.1, this UniFlash project can be included to enable an easier setup of the demonstration.

3.1.2.3.1 Programming Preconfigured Image Creator Project

1. Open UniFlash.
2. On the Choose Your Device section, select CC3220SF-LAUNCHXL, and make sure to select the *Serial* option and not *On-Chip* as shown in Figure 19. Click on the *Start Image Creator* button.

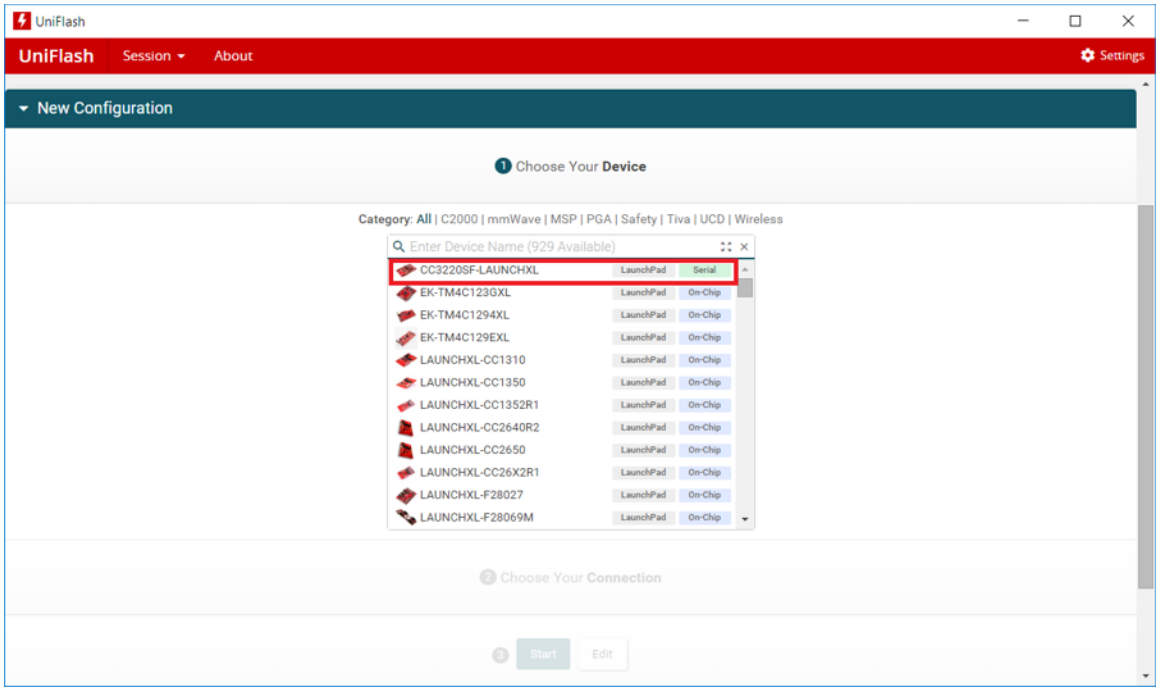


Figure 19. Choose Device: CC3220SF

3. After starting Image Creator, click on the *Manage Projects* button as shown in Figure 20.

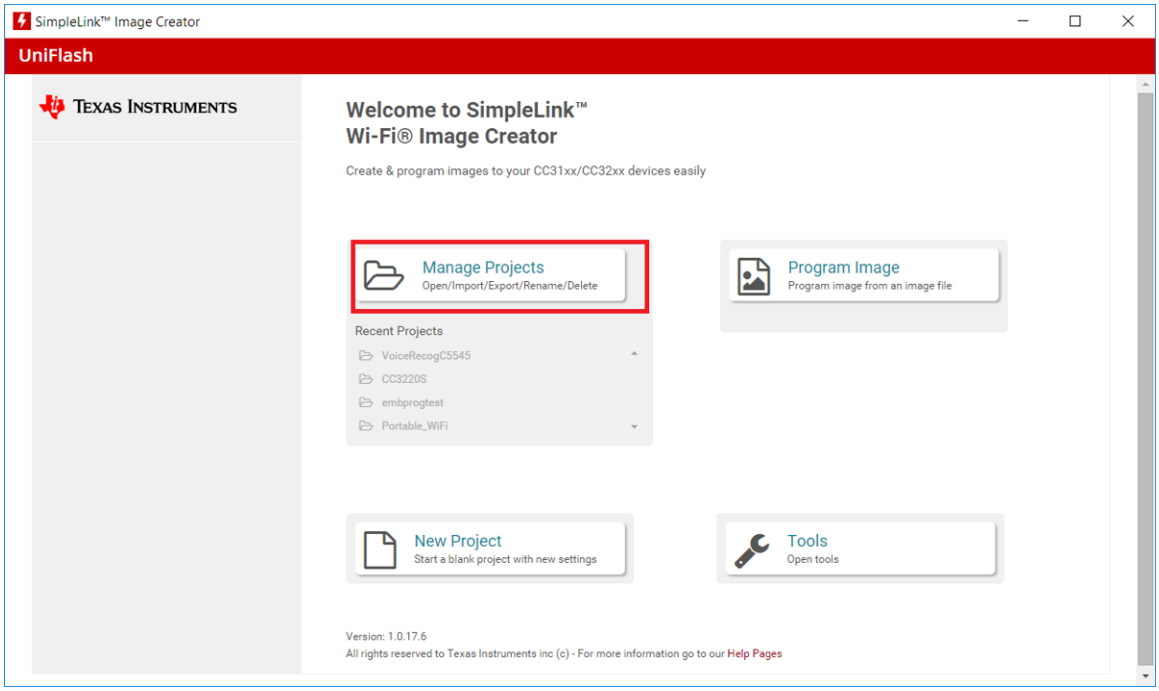


Figure 20. Manage Project

4. Click on the *Import Project from ZIP file* button, and select the zip folder <CSL INSTALL PATH>\c55_csl_x.xx\demos\voice_ui_cloud\cc3220. Select the appropriate zip file for the target device.

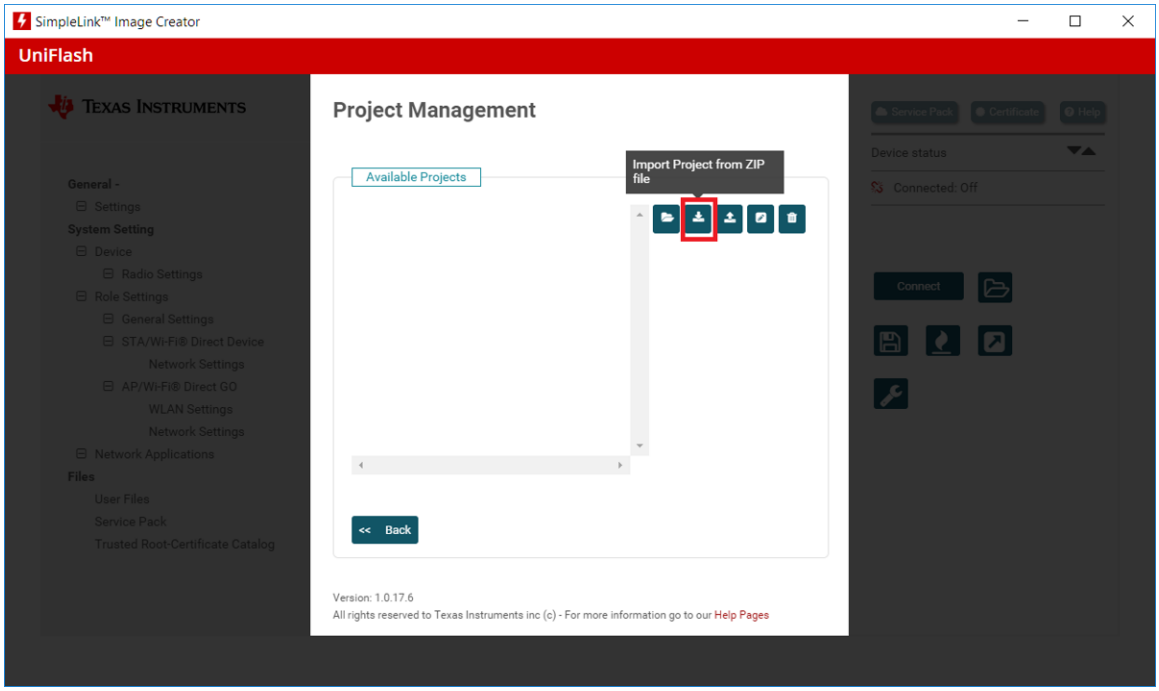


Figure 21. Import Project

5. Open the Voice Recog project.

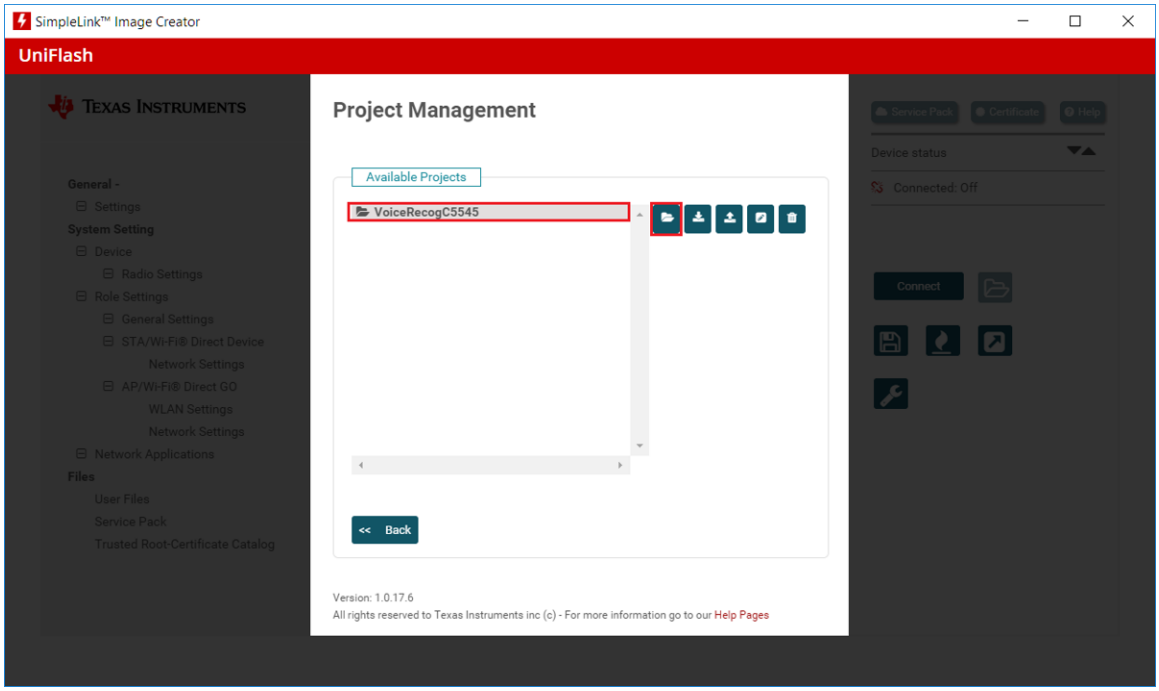


Figure 22. Open Project

6. Click on *User Files* in the left pane, and then click on the *Add File* button in the center pane. Navigate to the ibm_watson_key.txt file prepared in Section 3.1.2.4, and then click on *Write*.

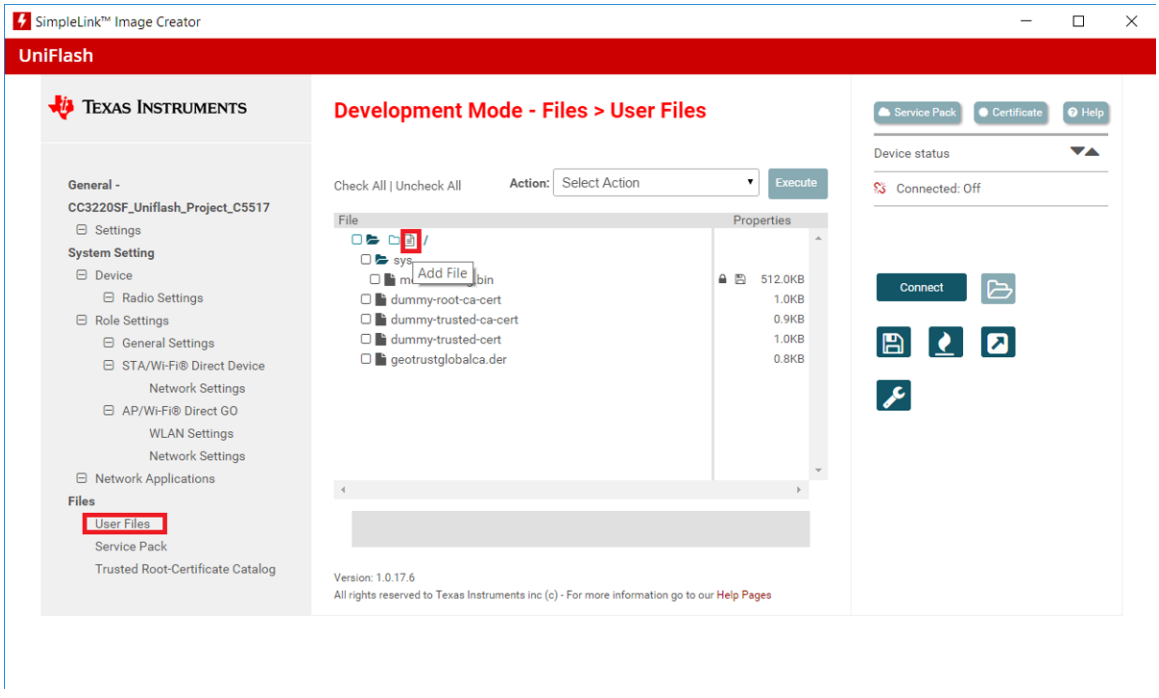


Figure 23. Add User File

7. Connect the device to the PC through a USB cable, and press the *Connect* button found on the bottom-right corner. Once the device is connected, select the *Generate Image* button underneath the Disconnect button, and select *Program Image (Create & Program)*.

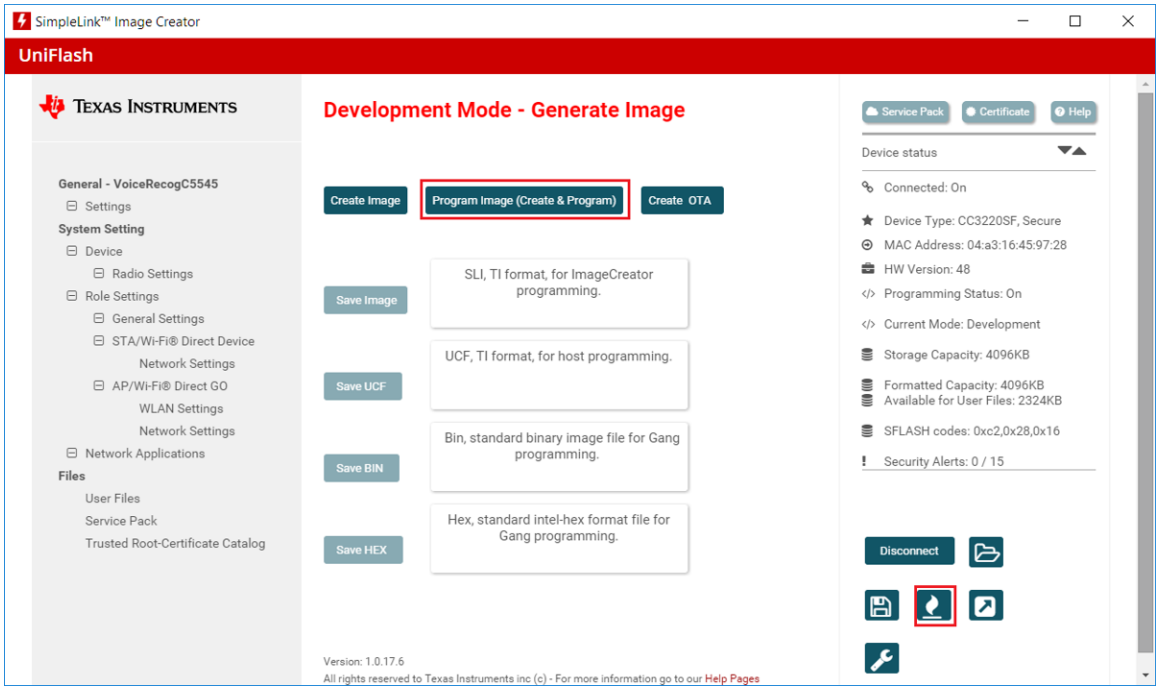


Figure 24. Generate Image

3.1.2.3.2 Creating an ImageCreator Project

The following steps allow the user to customize the example and use the new, updated files instead of the pre-built ones.

1. Open UniFlash.
2. On the Choose Your Device section, select CC3220SF-LAUNCHXL. Make sure the *Serial* option is selected and not *On-Chip* as shown in Figure 25. Click on the *Start Image Creator* button.

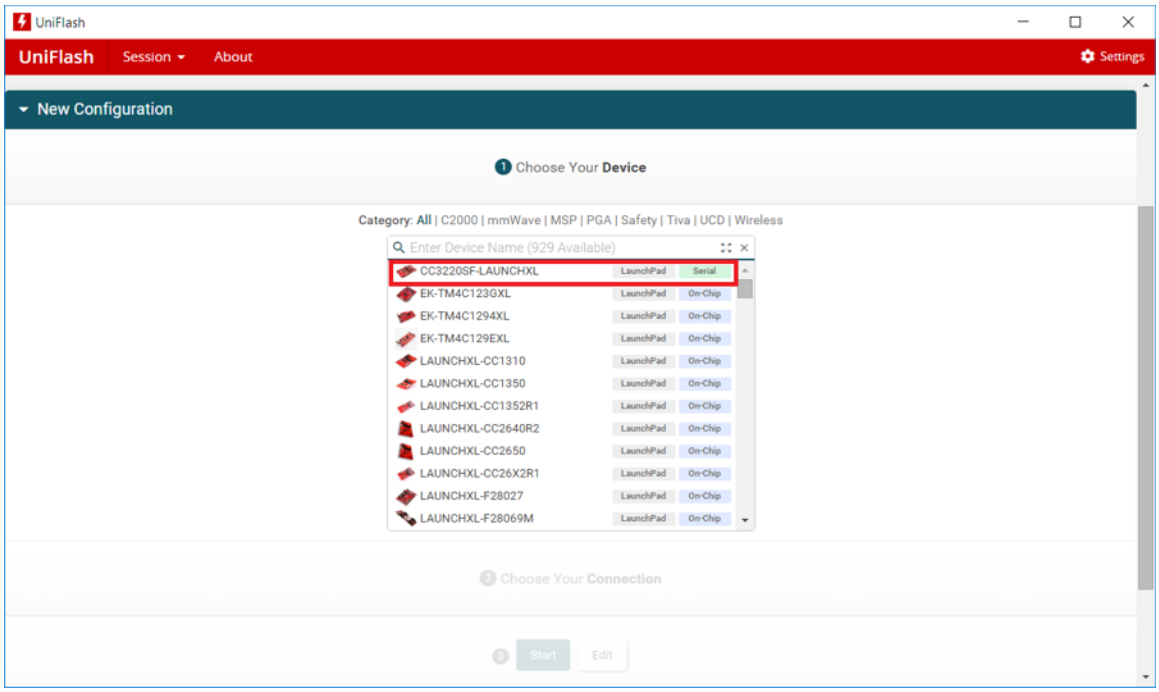


Figure 25. Choose Device: CC3220SF

3. After starting Image Creator, click on the *New Project* button as in Figure 26.

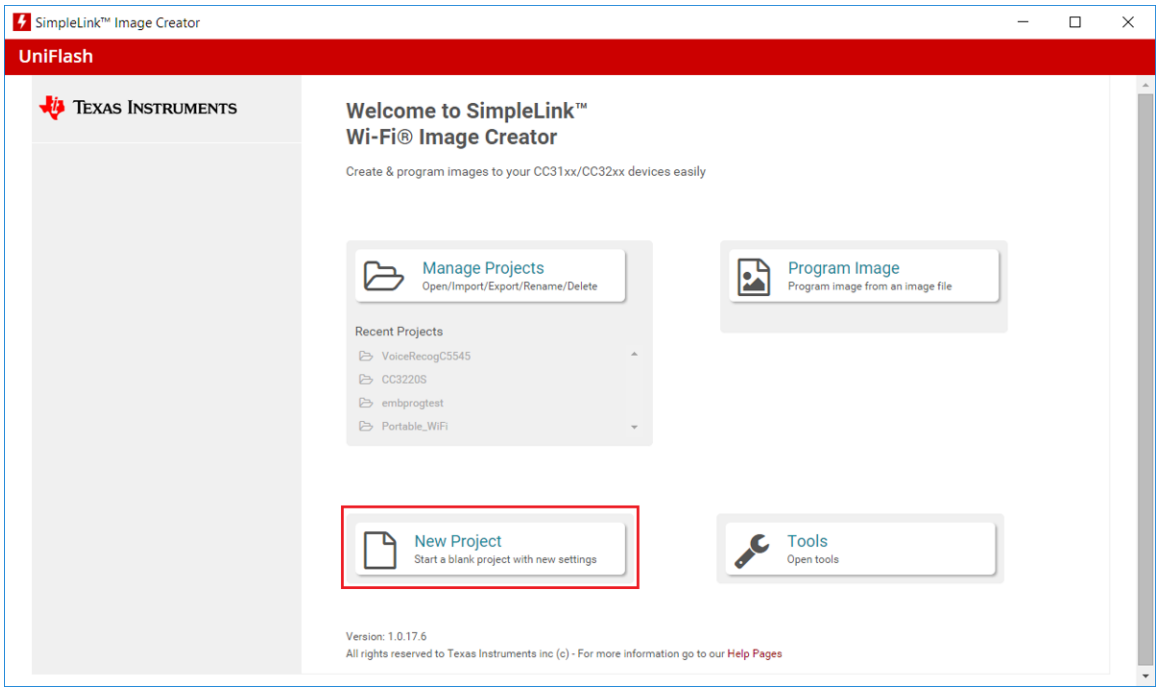


Figure 26. New Project

4. Enter a project name, select *CC3220SF* in the *Device Type* drop-down menu, make sure device mode is in *Develop*, and click on the *Create Project* button.

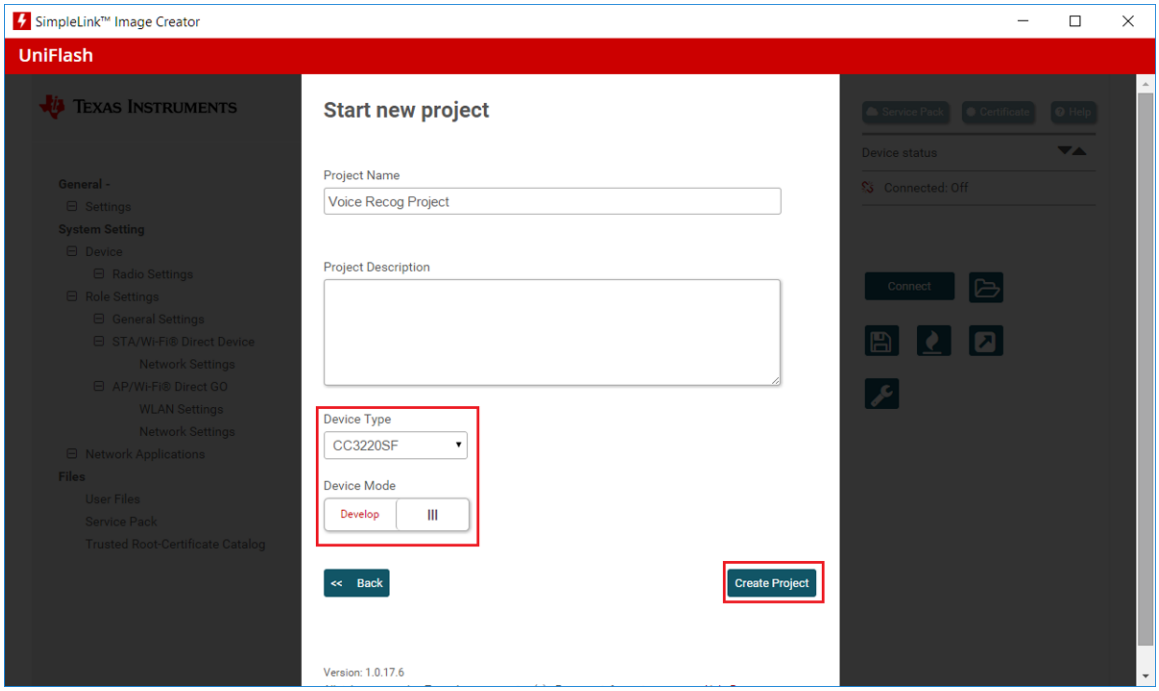


Figure 27. Starting a New Project

5. Select *Trusted Root-Certificate Catalog* in the bottom-left corner and uncheck the *Use default Trusted Root-Certificate Catalog* box. Include the *Source File* (certcatalogPlayGround20160911.lst) and *Signature Source File* (certcatalogPlayGround20160911.lst.signed.bin) found in <CC3220 SDK dir>\tools\cc32xx_tools\certificate-catalog.

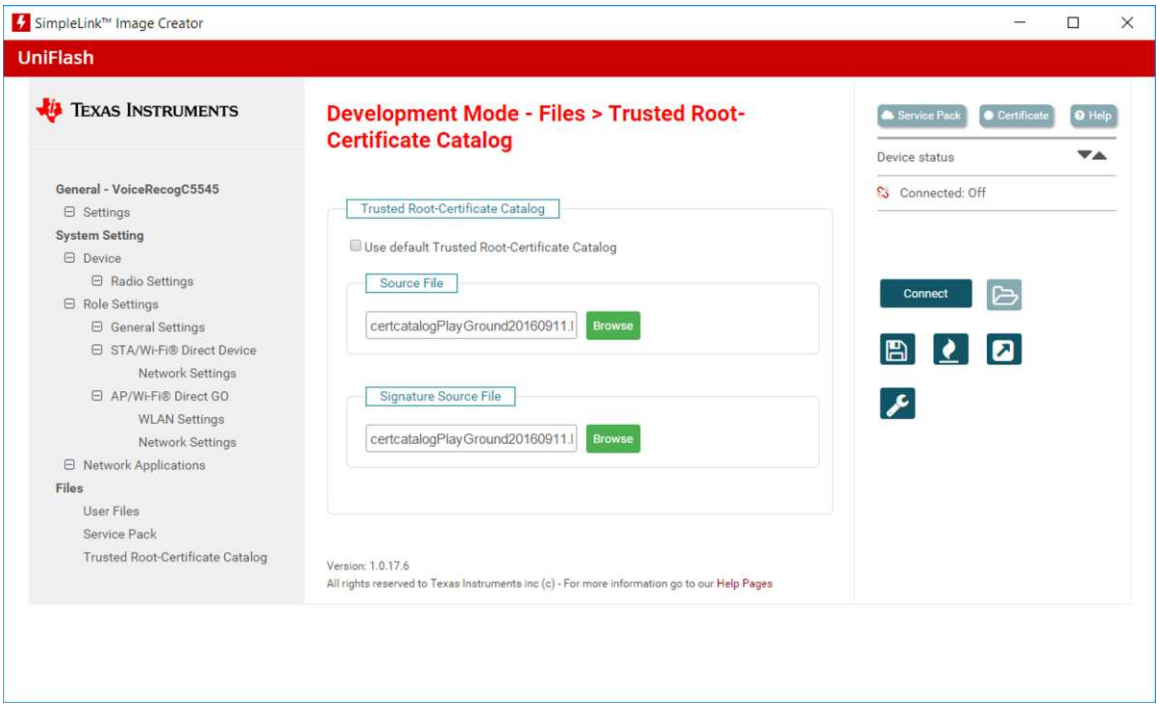


Figure 28. Trusted Root Certificates

6. Select *Service Pack* in the bottom-left corner and include the service pack bin (sp_3.3.0.0_2.0.0.0_2.2.0.4.bin) found in <CC3220 SDK dir>\tools\cc32xx_tools\servicepack-cc3x20.

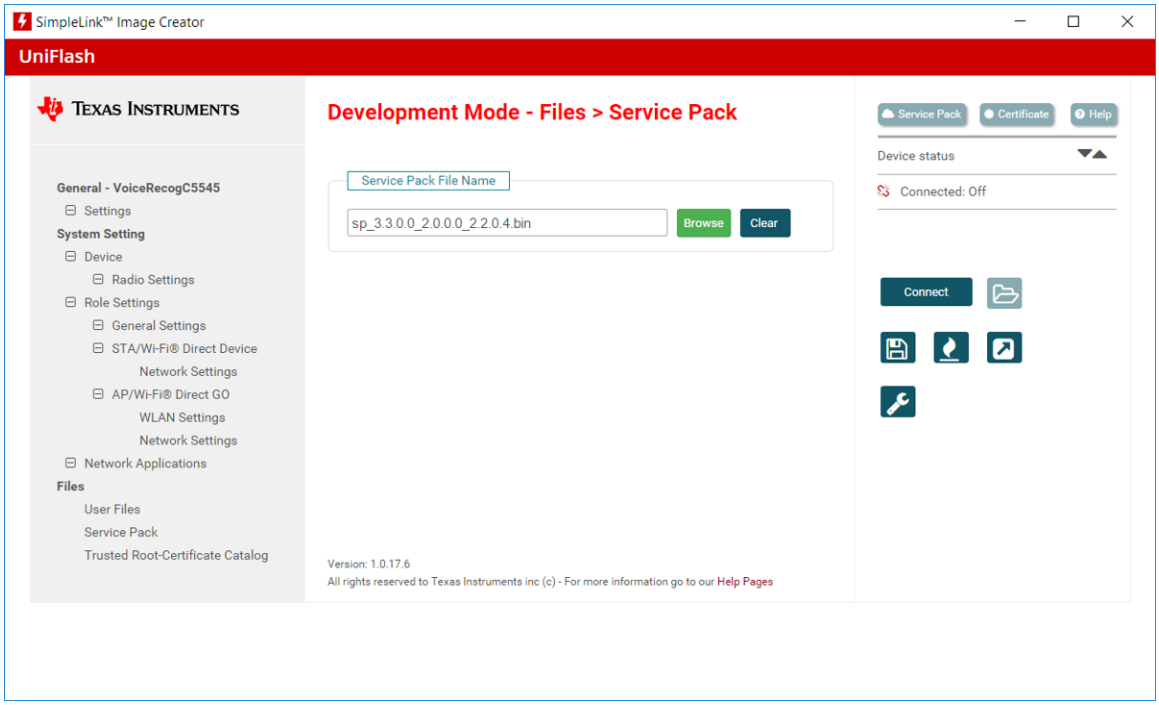


Figure 29. Service Pack

7. Select *User Files* and include the dummy-root-ca-cert, dummy-trusted-ca-cert, and dummy-trusted-cert files by clicking on the *Add File* icon. These files can be found in <CC3220 SDK dir>\tools\cc32xx_tools\certificate-playground. Leave any check boxes unchecked in the file add screen.

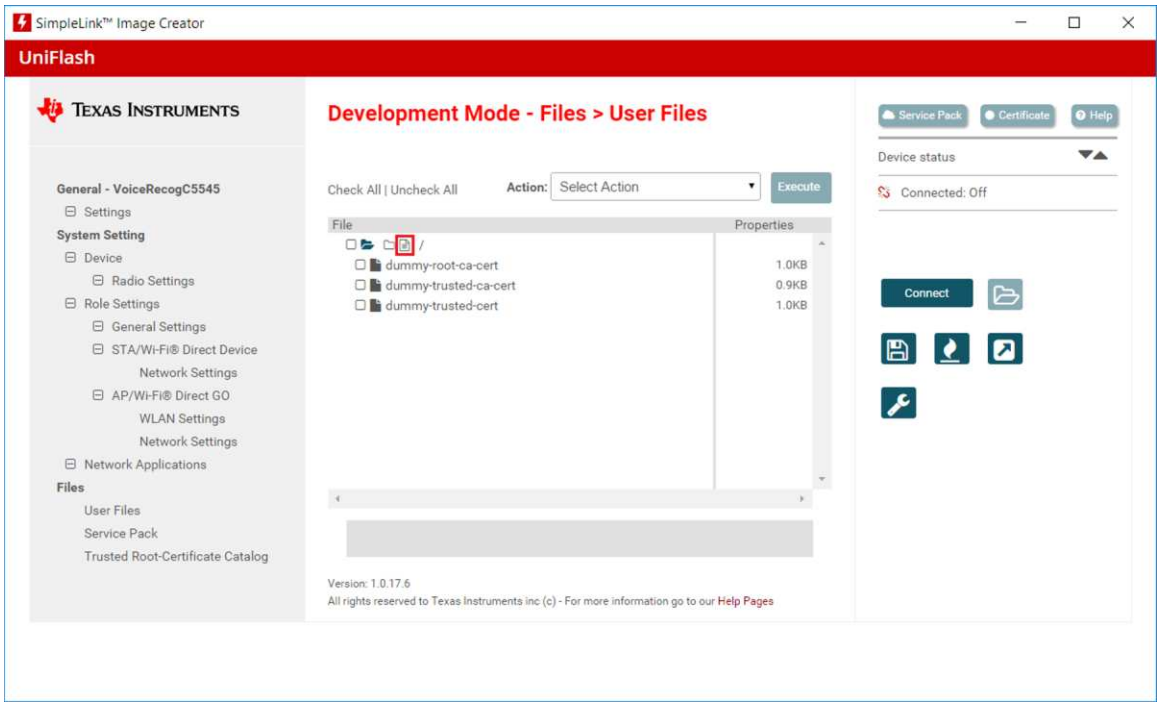


Figure 30. User Files

8. Add the geotrustglobalca.der file to the project by clicking on the *Add File* icon. That file can be found in <CSL INSTALL PATH>\c55_csl_x.xx\demos\voice_ui_cloud\cc3220\IBM_Voice_UI_Demo_CC3220SF_LAUNCHXL\geotrustglobalca.der.
9. On the drop-down box in the top-right corner, select *Select MCU Image*, and press *Browse*. On the next menu select *Private Key Name:* and include the dummy-trusted-cert-key (location <CC3220 SDK dir>\tools\cc32xx_tools\certificate-playground). On the *Certification File Name:*, select dummy-trusted-cert from the list. Ensure that the check boxes in Figure 31 are selected.

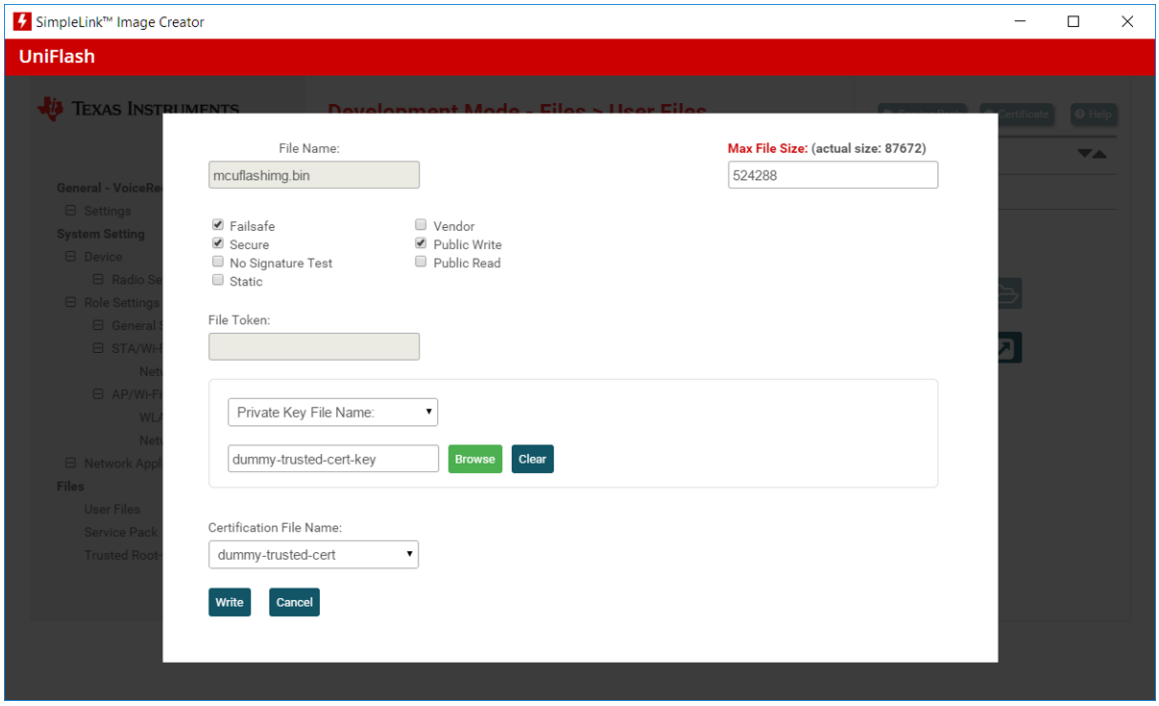


Figure 31. Select MCU Image

10. Add the ibm_watson_key.txt file to the project by clicking on the *Add File* icon. That file is prepared in Section 3.1.2.4.

11. Connect the device to the PC through a USB cable, and press the *Connect* button found on the bottom-right corner. Once the device is connected, select the *Generate Image* button underneath the *Disconnect* button. Select *Program Image (Create & Program)*.

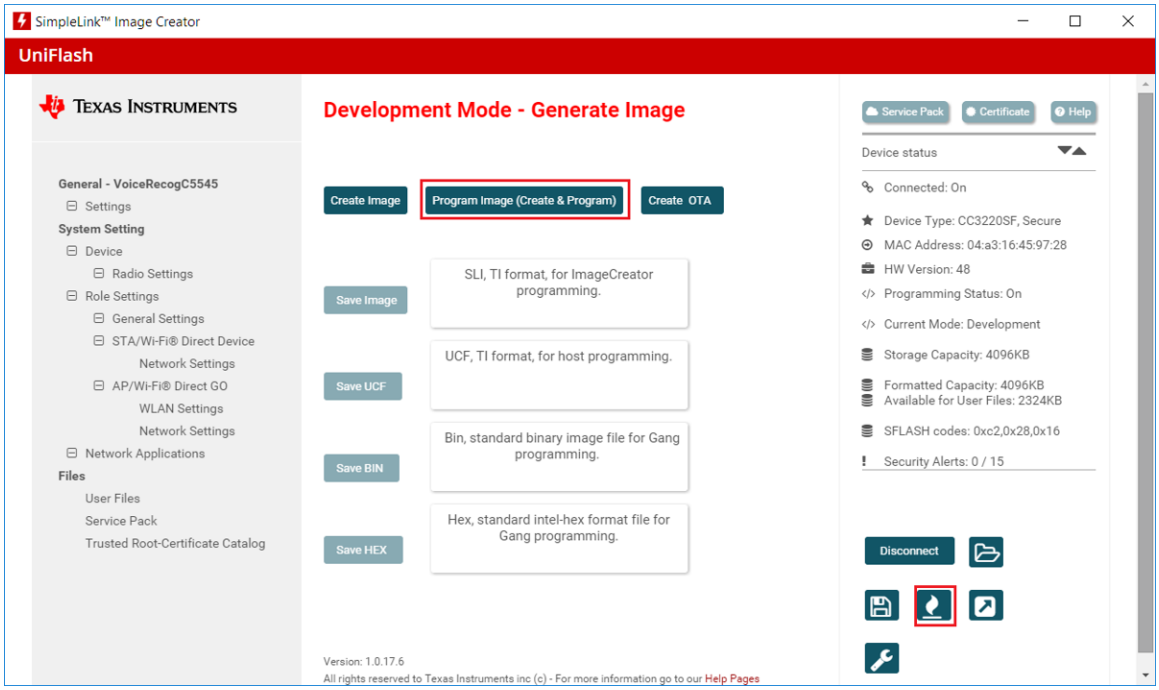


Figure 32. Generate Image

12. To verify that all files have been flashed correctly, open a terminal emulation session (Tera Term) and connect to the CC3220SF-LAUNCHXL COM port (XDS110 Class Application/User UART, BAUD 115200). After connecting to the emulation terminal, press the reset button on the CC3220SF-LAUNCHXL (SW1) and debug information prints as seen in Figure 33. It should indicate that the Watson API file is present and successfully connected to the Wi-Fi network.

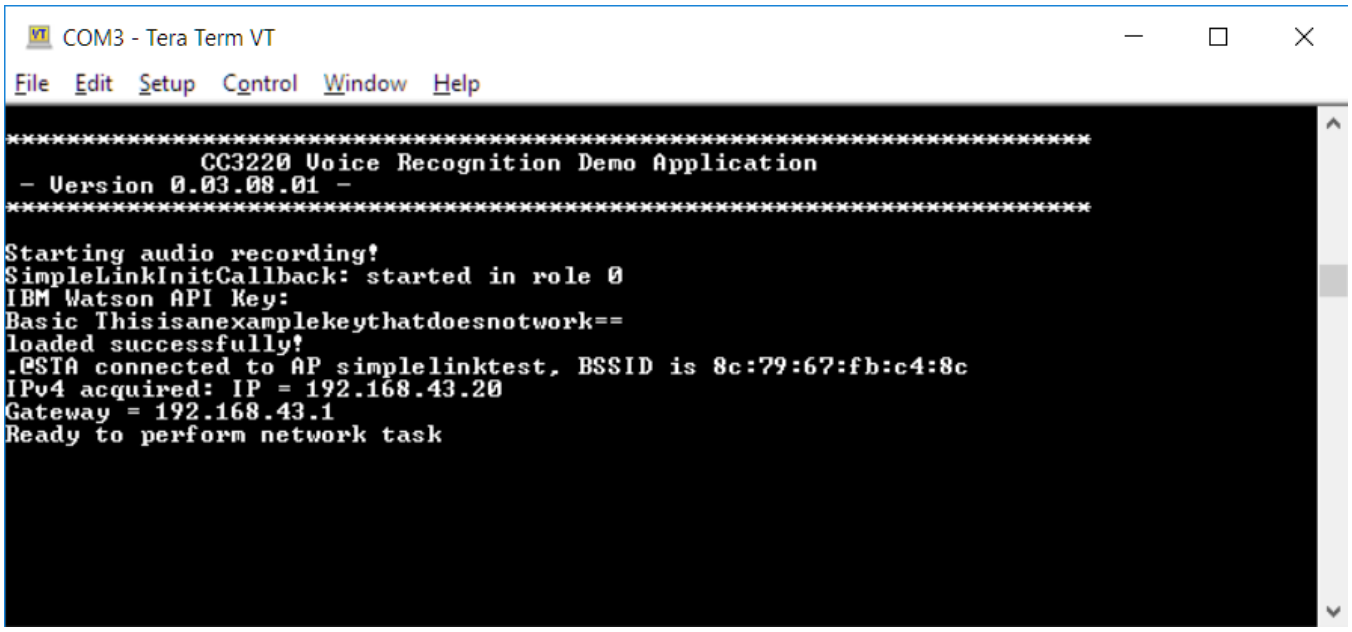


Figure 33. CC3220SF-LAUNCHXL Terminal Emulation Debug Information

3.1.2.4 IBM® Bluemix® Setup

This section covers how to create an IBM Bluemix account, enable Watson API access, and pass the Watson API keys to the CC3220 demonstration application.

1. To create an IBM Bluemix account, navigate to https://console.bluemix.net/ and sign up for a free account.
2. Confirm the Bluemix account using the link provided by IBM through email.
3. Log in to the Bluemix account, and walk through the steps on the screen.
4. Once finished with these steps, click on the *Catalog* tab located on the top-right corner.

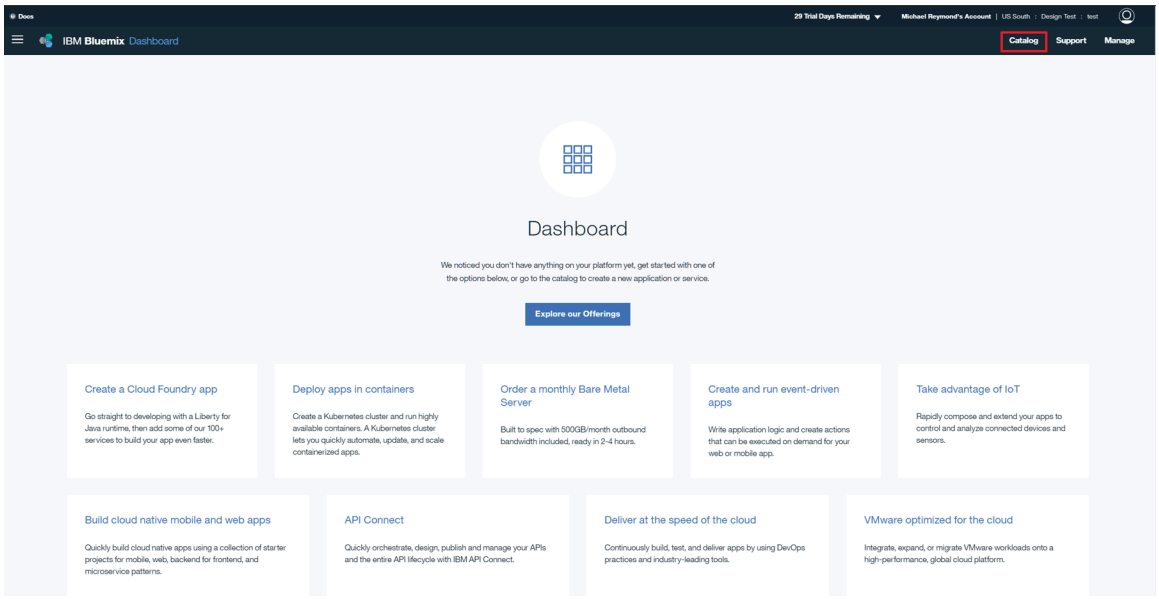


Figure 34. IBM® Bluemix® Dashboard

5. After selecting the *Catalog*, click on *Watson* under *Platform* on the left menu. Then, click on *Speech to Text*.

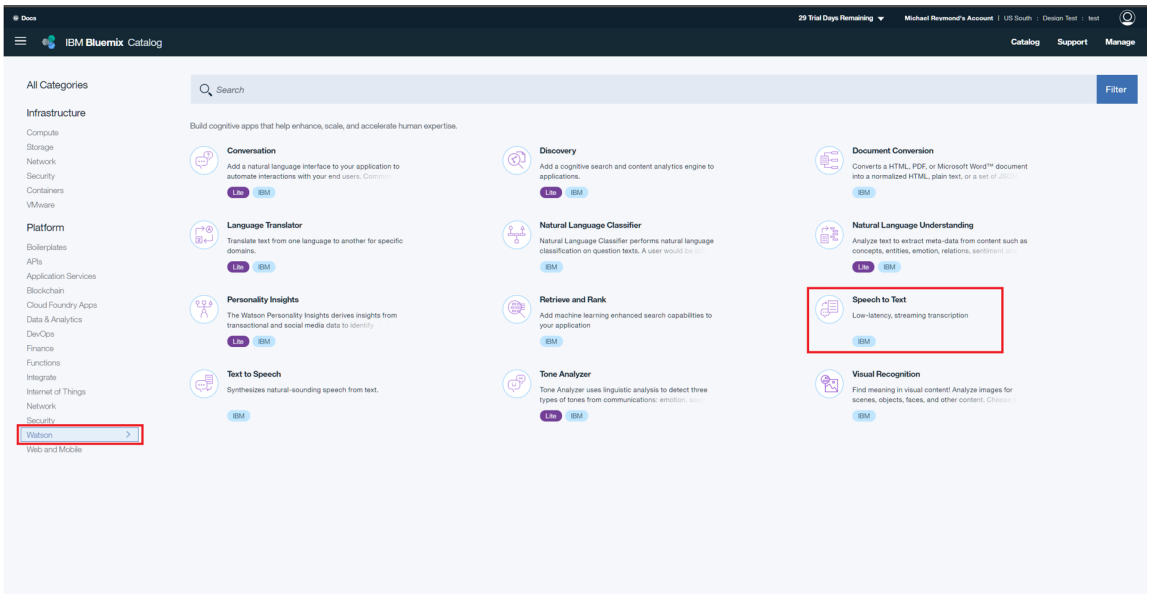


Figure 35. Watson™ Services

6. In the *Speech to Text* setup menu, leave everything as default, and then click on the *Create* button at the bottom right of the screen.

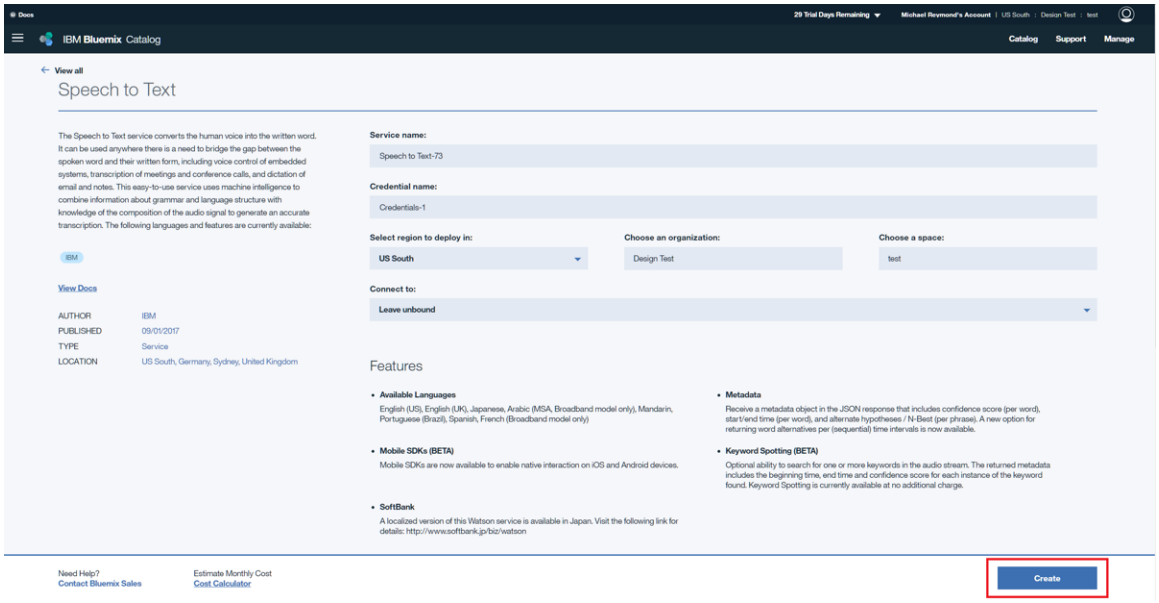


Figure 36. Create Watson™ Service

7. Once the service is created, the page refreshes and displays a management screen for the *Speech to Text* service. If the page does not refresh, then click on the IBM Bluemix logo to return to the main page, and then select the *Speech to Text* service.

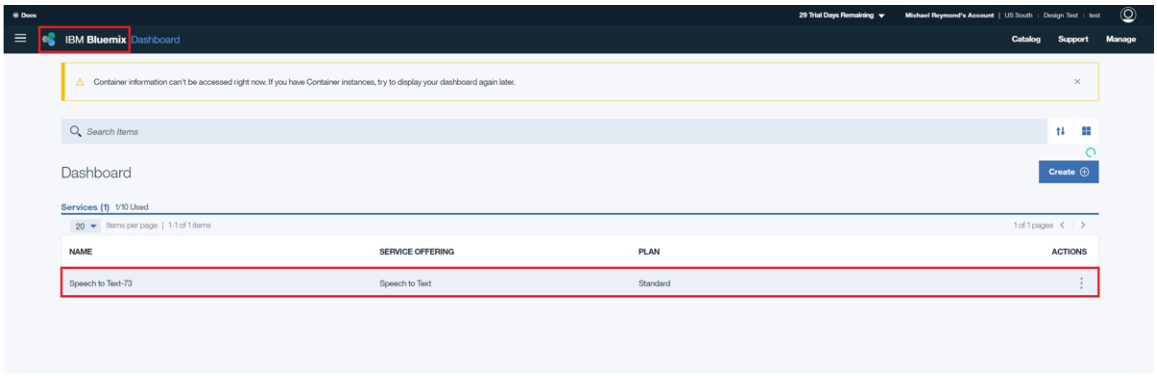


Figure 37. IBM ® Bluemix ® Dashboard With Speech-to-Text Service

8. Once in the management screen, click on *Service credentials*, and then click on *View credentials*.

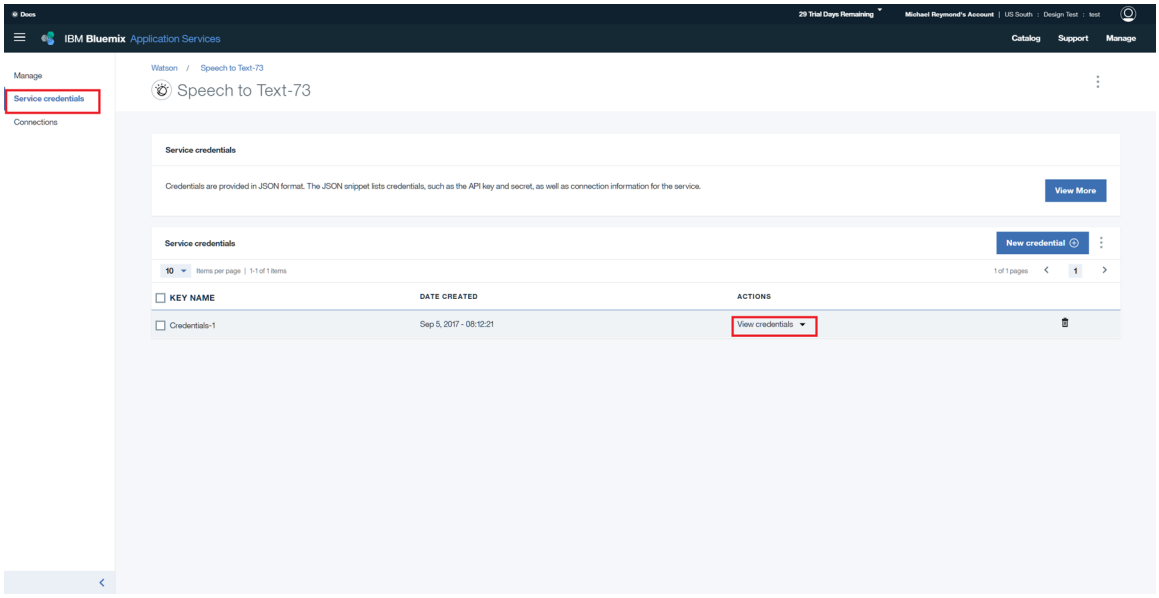


Figure 38. View Service Credentials

9. The new pane that appears once the *View credentials*is clicked on contains the Watson API authentication credentials. Now, convert the credentials into a hashed Basic Authorization token. To do this, concatenate the username and password with a colon separating them, encode the resulting string in Base64, and then prepend "Basic " to the Base64 encoded string. An automated way of performing the Base64 conversion can be found at http://decodebase64.com/. Assuming the following credentials from Watson:

```
{
"url": "https://stream.watsonplatform.net/speech-to-text/api",
"username": "myusername",
"password": "mypassword"
 }
```

10. The Base64 Encoding looks like Figure 39:

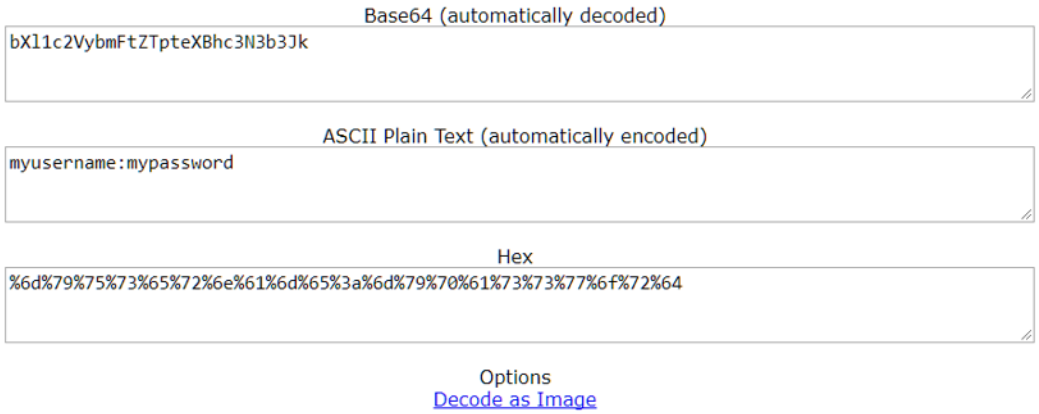


Figure 39. Base64 Encoding

11. Copy the output Base64 string in the upper-most text box and then prepend "Basic " to end up with a string that looks similar to the following example authorization token:
"Basic bXl1c2VybmFtZTpteXBhc3N3b3Jk"
Note that a real authorization token has a much larger length than the example presented here.
12. If using a precompiled binary, provide the application the API key in the form of an ibm_watson_key.txt file flashed onto the CC3220. To do this, open a preferred text editor (Notepad works for Windows® systems), create a new file, paste in the authorization token, and then save the file with the expected filename. An example (non-functional) key file is shown in ibm_watson_key_example.txt in the <CSL_INSTALL_PATH>\c55_csl_x.xx\demos\voice_ui_cloud\cc3220\ directory.

If compiling the binary without a precompiled binary, open src\httpvars.h and paste in the completed authorization token as the definition for AUTHENTICATION. This token allows the CC3220 example app to pass authentication credentials when accessing the Watson API over HTTP.

3.1.2.5 CC3220 Configuration Options

The CC3220 demonstration application is designed to be able to be built and run on the CC3220 without modifications. However, the demonstration application is very customizable and can be adapted to a wide variety of hardware and use cases. The configuration is done through an appconfig.h file and an httpvars.h file, which offer general application configuration options and HTTP-specific configuration options respectively.

3.1.2.6 App Configuration

In appConfig.h, there are a wide variety of configuration settings that can be set to certain values or turned on or off through the #define statements. The following is an explanation of each of the defines

- Application name defines
These defines are cosmetic and only affect the initial splash screen displayed on the console of a PC COM port emulator program connected to the CC3220 over the XDS110 USB debugger.
  - APPLICATION_NAME
This text is displayed on the console during the initialization of the application. Changing this define has no effect on the functionality of the application. Default: *CC3220 Voice Recognition Demo*
  - APPLICATION_VERSION
The version number of the program is displayed on the console during the initialization of the application. Changing this define has no effect on the functionality of the application. Default: *0.03.08.01*
- Ping parameters defines:
These defines control the behavior of the ping function, which is unused in the application, but can be used in the program for network troubleshooting purposes.
  - PING_INTERVAL
This controls the amount of time between two successive ping requests, in milliseconds. Default: 1000
  - PING_TIMEOUT
This controls the amount of time the network processor waits for a ping response before it times out and marks the request as failed. Default: 3000
  - PING_PKT_SIZE
This controls how much is data is sent in a ping request, in bytes. Default: 20
  - NO_OF_ATTEMPTS
This controls how many ping requests the NWP will send. Default: 3
- Wi-Fi configuration defines:
These defines control the behavior of the CC3220's Wi-Fi connection to an access point
  - USE_PROVISIONING
This controls whether the application uses the provisioning task to provision to an access point if it cannot connect using any saved Wi-Fi profiles. If provisioning is disabled, then the static Wi-Fi parameters defined here are used to connect to the Wi-Fi access point. If provisioning is enabled, then the SimpleLink Starter Pro app must be used to provision the device.

Default: 0

- SSID_NAME
  This define is used as the SSID of the access point that the CC3220 attempts to connect to if provisioning is disabled. Must change this define to match the SSID broadcast by a nearby AP. Default: *simplelinktest*
- SECURITY_TYPE
  If the AP broadcasting the SSID defined in SSID_NAME has security enabled, then this define must be set to the match the security type of the AP. Default: SL_WLAN_SEC_TYPE_WPA_WPA2
- SECURITY_KEY
  If the AP broadcasting the SSID defined in SSID_NAME has security enabled, then this define must be set to the security key of the secured AP. Default: “wifitest”
- SSID_LEN_MAX
  This define controls the size of the char array used to store the SSID_NAME. Must be defined as at least the size of SSID_NAME. Default: 32
- BSSID_LEN_MAX
  This define controls the size of the char array used to store the BSSID of the AP. Must be defined as at least 6 (the length of a MAC address). Default: 6
- AYNC_EVT_TIMEOUT
  This define controls the timeout used by the provisioning task, in milliseconds. Default: 5000
- SL_STOP_TIMEOUT
  This define controls the timeout used by sl_stop(), in milliseconds. Default: 200

- Audio input configuration defines:
  These defines control the audio input source. While the C55X software only supports connecting to the CC3220 using UART1, the CC3220 demonstration application supports other audio inputs, as long as they provide 16-bit PCM audio at 16KHz. Note that only one input is allowed at a time, and only one of the USE_X_AUDIO_IN defines can be set to 1.
  - USE_ANALOG_AUDIO_IN
    This define controls whether the audio input is received through the ADC. When enabled, analog audio must be input to p60, and the application will sample the ADC at 16 kHz for the audio input. Default: 0
  - USE_I2S_AUDIO_IN
    This define controls whether the I2S interface is used to capture audio. When enabled, 16-bit PCM audio is read from the I2S interface. This audio must be given as a two-channel, 16-kHz stream. Default: 0
  - USE_UART_AUDIO_IN
    This define controls whether the UART interface is used to capture audio. When enabled, data is read from the UART interface and used as the audio input. This audio must be given as a one-channel, 16-bit, 16-kHz stream. Default: 1
  - USE_UART0
    This define controls whether the UART0 interface is used for audio capture, instead of UART1. This is mostly useful only for debugging, as UART0 is used for the PC COM port. With audio input from UART0, it is possible to use a PC terminal program to feed arbitrary test audio data to evaluate the performance of the speech recognition and transcription service. Default: 0
- UART-input specific defines:
  These defines control some UART-specific behavior. All defines except UART_BAUDRATE must be set to 0 if UART audio input is not used.
  - UART_BAUDRATE
    This define controls the baud rate of the UART1 interface. This must be set to the same baud rate used by the C55X. Default: 1,228,800
  - UART_SYNC_IN_USE
    This define controls whether a sync word is expected in the audio input. As UART is asynchronous, a sync mechanism is required to detect data corruption. If enabled, the CC3220 application checks the first 4 bytes of an audio frame and discards the frame if *SYNC* is not present. This must be enabled if the audio input is a C55X running the demonstration application. This must be disabled if the sync is word is not used, such as if any non-UART input is used. Default: 1

  - USE_CONTINUOUS_UART_RECORDING
    This define controls whether the UART1 interface is always on. With this define enabled, the UART1 interface is opened during application initialization, and kept open and listening at all times. This allows for the CC3220 to not need an external trigger source to prompt it to start recording audio data, as any UART data received will function as the trigger. This assumes that the audio input device only sends UART data once it has been triggered through user interaction. This must be enabled if the audio input is a C55X running the demonstration application. This must be disabled if UART is not in use, or if the UART audio input device sends UART data continuously. Default: 1
  - TIMEOUT_TICK_COUNTS
    This define controls how long the application waits for the UART audio input device to finish sending audio data once it has been triggered, in tenths of a second. This must be set to a value slightly larger than the maximum recording time defined in MAX_FRAMES below. Default: 50

- I2S-specific defines:
  These defines control some I2S-specific defines. All defines must be set to 0 if I2S audio input is not used.
  - CONFIG_AUDIO_BOOSTERPACK
    The I2S input is tested with a CC3200 audio BoosterPack, which features an onboard AIC3254 codec. For the codec to output the expected 16-bit, 16-kHz I2S audio, configuration is needed. If this define is enabled, then during audio initialization, the ConfigureAudioCodec() function is called, which configures all of the AIC3254 registers to their appropriate values using I2C. To support another codec, a user-supplied version of ConfigureAudioCodec() would have to be used. This define must be disabled if the codec used is not an AIC3254, or if I2S audio input is not used. Default: 0
  - USE_2CH_AUDIO
    This define controls whether the received I2S data is treated as a two-channel audio stream by the application in the communication with the IBM Watson service. This does not change the actual functionality of the I2S interface, and the PCM audio data must be packed as a two-channel stream even if the audio data is only one channel. This must be enabled if the I2S data input is composed of two channels. This must be disabled if the I2S data input is one channel audio, or if I2S is not used. Default: 0
  - BUGGED_I2S_CALLBACK
    This define controls whether the software workaround for the I2S callback bug is fixed. In old (<1.40) SDK versions, the CC3220 I2S driver has a bug where the callback for a I2S read call is called when a buffer starts reading data, and not when it is done reading data. This must be enabled if using a CC3220 SDK version earlier than 1.40.00.03. This must be disabled if using CC3220 SDK version 1.40.00.03 or later. Default: 0

- General application config defines:
  These defines control general application behavior.
  - ENABLE_DEBUG_OUTPUT
    This define controls whether the CC3220 application displays verbose UART0 output to the PC COM port. Changing this define has no effect to the functionality of the application. Default: 1
  - ENABLE_C5545_PIN_SETTINGS
    This define controls whether the CC3220 has its pin mux settings configured to support interfacing with a C5545 BoosterPack using direct connection with the LaunchPad headers. Enabling this define disables the Sharp display output, disables the LEDs, and muxes out the UART pins to p8 for UART RX and p7 for UART TX. This define must be enabled if using the CC3220 demonstration with the C5545 BoosterPack demonstration application. This must be disabled otherwise. If this define is enabled, USE_UART_AUDIO_IN and USE_CONTINOUS_RECORDING must be also enabled. If this define is enabled, USE_ANALOG_AUDIO_IN, USE_I2S_AUDIO_IN, and USE_UART0 must not be enabled. Default: 1
  - MAX_TRANSCRIPT_LENGTH
    This define controls how many characters the CC3220 saves from the returned IBM transcript and how many characters the CC3220 sends to the C5545 for display on the OLED. Default: 100
  - MAX_FRAMES
    This define controls the size of the audio recording buffer. This is defined in 160 sample frames (320 bytes). Each frame represents 10ms of audio at 16kHz, so this define effectively sets the

recording length in hundredths of a second. Default: 450

3.1.2.7 HTTP Configuration

In httpvars.h, there are some configuration settings relating to the HTTP transfer to the IBM Watson server. The following is an explanation of each of the defines:

- POST_REQUEST_URI
  This define controls the URI used to send the POST request once connected to the IBM Watson server. Do not change this unless the Watson API changes.
  Default: */speech-to-text/api/v1/ recognize?smart_formatting=true&max_alternatives=1*
- HOST_NAME
  This define controls the host name that the CC3220 connects to. Do not change this unless the Watson API changes.
  Default: *stream.watsonplatform.net*
- HOST_PORT
  This define controls the host port that the CC3220 connects to. Do not change this unless the Watson API changes.
  Default: 443
- USE_PROXY
  This define controls whether a HTTP proxy is used to tunnel HTTP traffic.
  Default: 0
- PROXY_IP
  This define controls the IP address of the HTTP proxy server.
  Default: <none>
- PROXY_PORT
  This define controls the port of the HTTP proxy server.
  Default: <none>
- MAX_BUFF_SIZE
  This define controls the size of the buffer used to store HTTP data received back from the IBM Watson server.
  Default: 1460
- FILE_NAME
  This define controls the filename of the test audio file. As a debug feature, it is possible to use the contents of a prerecorded audio file as the input to the IBM Watson server. In this case, a .flac file must be transferred to the CC3220 using Uniflash, and its filename must be provided in this define.
  Default: *audio-file.flac*
- CONTENT_TYPE
  This define controls the HTTP content-type header value passed in the HTTP POST requests when an audio file is used as input. For a flac file, this must be defined as “audio/flac”. Do not change this unless the Watson API changes.
  Default: *audio/flac*
- CONTENT_TYPE_RAW
  This define controls the HTTP content-type header value passed in the HTTP POST requests when a streaming audio input is used. Do not change this unless the Watson API changes.
  Default: *audio/l16;rate=16000;channels=1*
- CONTENT_TYPE_RAW_2CH
  This define controls the HTTP content-type header value passed in the HTTP POST requests when a streaming audio input is used and a two-channel audio source is used. Do not change this unless the Watson API changes.
  Default: audio/l16;rate=16000;channels=2
- API_KEY_FILENAME
  This define determines the filename of the IBM Watson API key file. The application reads this file and uses its contents as the API key if READ_API_KEY_FROM_FILE is set.
  Default: *ibm_watson_key.txt*
- READ_API_KEY_FROM_FILE
  This define determines whether the IBM Watson API Key is read from the API_KEY_FILENAME file or

whether the API key defined in AUTHENTICATION is used. Set to 1 to read from file. Set to 0 to use AUTHENTICATION define. If set to 1, the user must provide valid key file. If set to 0, the user must provide valid key in AUTHENTICATION.
Default: 1

- AUTHENTICATION
  This define controls the authentication key used when connecting to the IBM Watson service. This key is a user-specific key that has format of *Basic <69 bytes>*. This key must be provided and defined using the procedure described in Section 3.1.2.4.
  Default: <blank>
- ROOT_CA_CERT
  This define controls the root certificate authority certificate that the IBM Watson server's certificate is signed against. This define must be provided for the TLS handshake to complete without security warnings, as this certificate is needed to validate the IBM Watson certificate chain. Do not change this unless the IBM Watson server changes its certificate chain.
  Default: *geotrustglobalca.der*
- DEVICE_YEAR

  This define sets the device year. The application uses this define to set its internal date and verify that the server's certificate has not expired. If the define is not set, a security error will result. This define must be set to the current year.

  Default: 2017
- DEVICE_MONTH

  This define sets the device month. The application uses this define to set its internal date and verify that the server's certificate has not expired. If the define is not set, a security error will result. This define must be set to the current month.

  Default: 8
- DEVICE_DATE

  This define sets the device day. The application uses this define to set its internal date and verify that the server's certificate has not expired. If the define is not set, a security error will result. This define must be set to the current day.

  Default: 30

3.1.2.8 C55x Configuration Options

3.1.2.8.1 Changing Filter Coefficients

The BF filter coefficients depend on the geometry of the microphone array. The filter coefficients in the this project are calculated based on a four-microphone geometry of the LMB board. If the user wishes to use a different microphone array of a different geometry, new filter coefficients are required. Section 3.1.2.8.1.1 describes how to calculate a new set of filter coefficients for the geometry of the microphone array. The new filters' coefficient buffers are updated and the project must be rebuilt. The file sysbffilt.c has the values of the filters. The file sysbffilt.h is the include file associated with the filters. Both files are located in c55xx_csl\demos\audio-preprocessing\common subdirectory.

3.1.2.8.1.1 Calculating Filter Coefficients

The BF filter coefficients depend on the geometry of the microphone array and the angle of the direction of the source with respect to the microphone array. bfgui.exe is a tool to generate BF filter coefficients and is part of the AER library in directory *aer_c55l_cpuv3.3_obj_17_0_0_0\tools\bf_tool*. A user's guide for the BF design tool bfgui.pdf is in the same directory as well. The user is strongly encouraged to read bfgui.pdf because it gives insight into the general theory of BF.

Upon starting bfgui.exe, the user should configure the following values, as shown in Table 11.

Table 11. BF Design Tool Values

VALUE	COMMENTS
Sampling rate (kHz)	This reference design uses 16 (16000). The default value of the tool is 8000.
Number of microphones	Two or four microphones are used.
Microphone distance	Distance in centimeters between two adjacent microphones. Equal distance between any two microphones is assumed. For linear array, it is the linear distance, and for circular array, it is the linear distance of the chord between two microphones.
BF angle	The utility generates a set of filters for a single angle of arrival; that is, for a single virtual-directional microphone. For a system with multiple virtual-directional microphones like the one that is used in this TI Design, the utility must be called multiple times, each time for a different angle.
Geometry	Microphone array geometry: 0 for 1D linear, 1 for 2D linear, 2 for 2D rectangular, and 3 for circular. Using the LMB requires geometry be set to 0.
Contour levels	Required for graphical illustration. Leave as default.
Polar frequency	Required for graphical illustration. Leave as default.

The number of microphones on the array determines the uniqueness of the separation. The number of virtual-directional microphones determines the angle of separation. If the number of virtual microphones was set to twelve, then every 30° (360° divided by 12) is a virtual-directional microphone. There is a relationship between the number of microphones and the number of virtual-directional microphones. The processing load of the BF and the ASNR depends on the number of microphones in the array and linearly on the number of virtual-directional microphones. Benchmark results for typical C5517 DSP are given in section.

Following the instructions in the user's guide (bfgui.pdf), configure the filter coefficients tool for linear geometry and four microphones with 2.125-cm equal distance between any two microphones. This filter is for 45° of arrival.

Figure 40 shows the configuration of the filter generation tool. The filter coefficients are stored in filterCoeff.log.

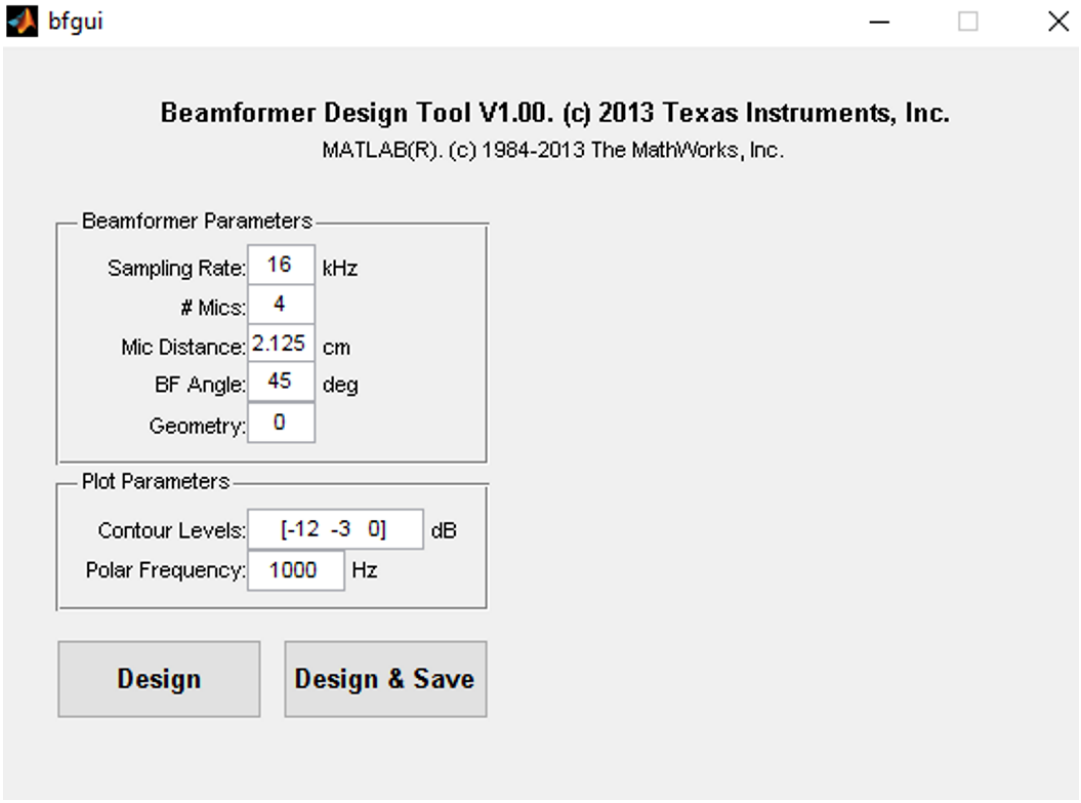


Figure 40. Configuration of Filter Generation Tool

Figure 41 illustrates a virtual microphone.

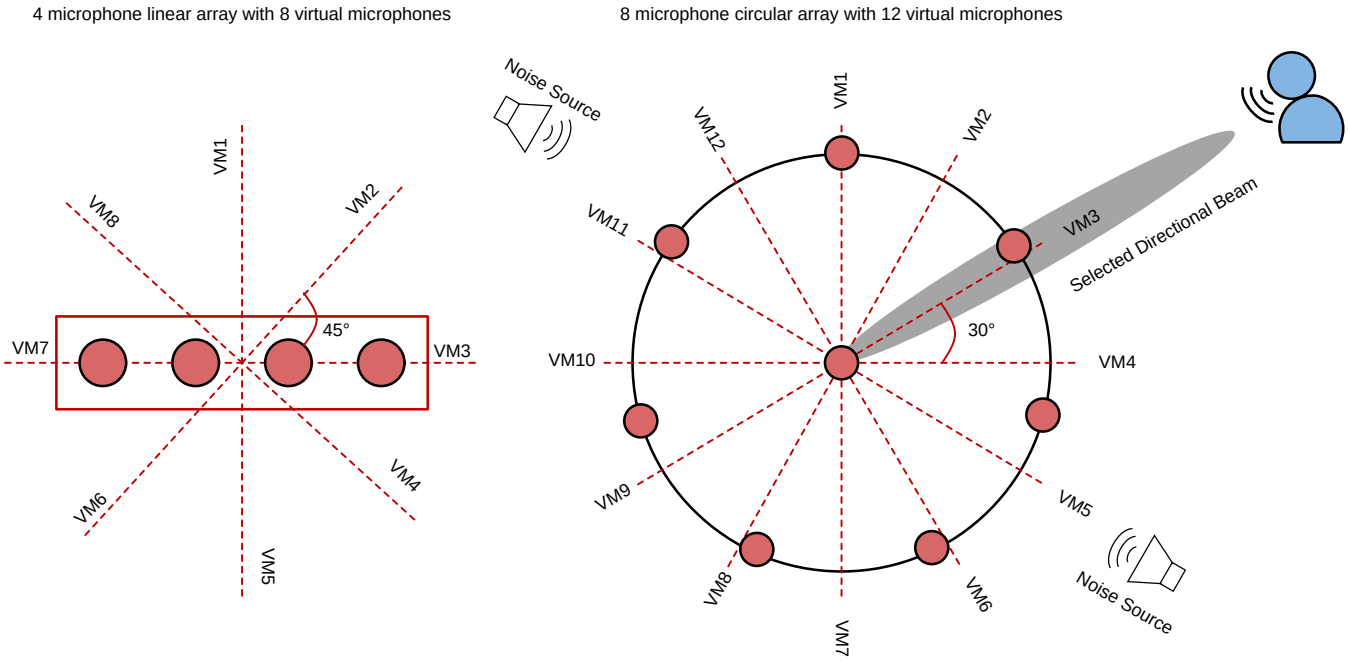


Figure 41. Description of Virtual Microphone

3.2 Testing and Results

3.2.1 Test Setup

This section covers how to run the two demonstrations. As previously mentioned, each demonstration runs independently and allows the user to evaluate the C5517 and C5545 with the IBM Watson voice transcription service.

3.2.1.1 TMDSEVM5517 and CC3220SF-LAUNCHXL Demonstration

This section assumes the following:

- The hardware is connected as described in Section 3.1.1.1.1.
- CCS has been installed and launched.
- The set of tools and utilities from Section 2.2.12 have been installed on the PC. The directory structure with the dependencies must look like Figure 42 and Figure 43.

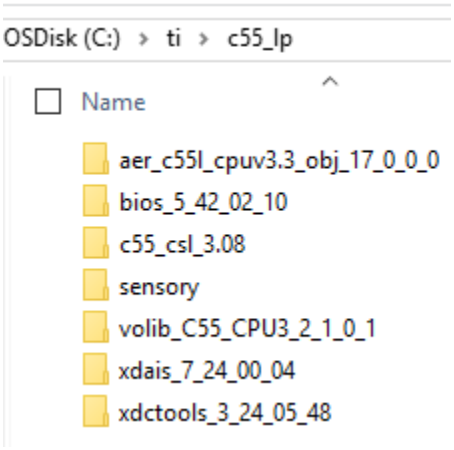


Figure 42. Directory Structure of Project Dependencies and CSL Installation on PC

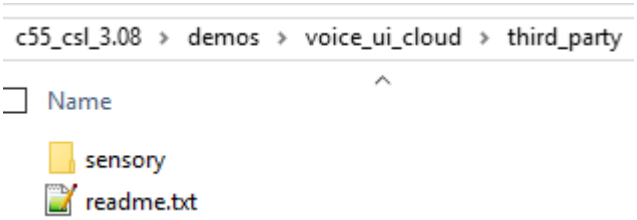


Figure 43. Location of Sensory Dependency Installation

- A target configuration for TMDSEVM5517 has been created (not covered in this document).
- A target configuration for CC3220SF-LAUNCHXL has been created (not covered in this document).
- Ensure the CC3220SF-LAUNCHXL is powered and its binary flashed as described in Section 3.1.2.3. The CC3220SF-LAUNCHXL must also be connected to a Wi-Fi hotspot with an Internet connection.

NOTE: A pre-built binary that can be loaded and run through CCS is provided in the C55x CSL package at *<CSL ROOT>\demos\voice_ui_cloud\c5517\Debug\c5517_voiceuicloud.out*.

If a bootable SD card image is required, the image is in the same location and is named *bootimg.bin*. For instructions on creating and booting from an SD card on the EVMC5517, see Section 6.

1. With CCS launched, import the C5517 project and C55XXCSL_LP into the workspace from *<CSL INSTALL ROOT>\demos\voice_ui_cloud\c5517* and *<CSL INSTALL ROOT>\ccs_v6.x_examples*

respectively. The workspace would look like Figure 44.

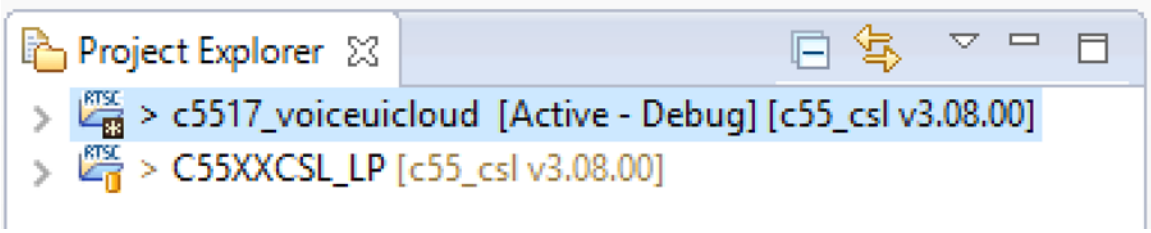


Figure 44. TMDSEVM5517 and CC3220SF-LAUNCHXL Demonstration CCS Workspace

2. Prior to building the demonstration, ensure csl_general in C55XXCSL_LP has the correct macro for the C5517 #define CHIP_C5517. The file is located at *<CSL ROOT>\inc*. Once the correct macro is defined, rebuild the C55XXCSL_LP project.
3. Right-click on the c5517_voiceuicloud project, and click build project.
4. When the build completes, c5517_voiceuicloud.out will be present in the *Debug* folder.
5. Power on the EVM5517 and connect a USB cable to the onboard debugger J3.
6. Ensure the CC3220SF-LAUNCHXL is powered and its binary flashed as described in Section 3.1.2.3. The CC3220SF-LAUNCHXL must also be connected to a Wi-Fi hotspot with an Internet connection. If desired, the CC3220SF-LAUNCHXL debug information can be viewed in a terminal emulation software, such as TeraTerm. See step 12 of Section 3.1.2.3.2 for details on connecting to the CC3220SF-LAUNCHXL terminal window.
7. Launch the target configuration for the EVM5517.
8. Right-click and connect to the DSP core with a GEL file loaded.
9. Load c5517_voiceuicloud.out onto the core.
10. Press resume, and there will be debug information from the initialization of the LMB printed on the CCS console. A blue LED on the LMB will illuminate, which indicates a successful initialization of the onboard PCM1864.
11. At this point the demonstration is ready to run. The audio captured by the LMB can also be heard through headphones connected to P9 (green connector) on the EVM5517. The processed (put through BF, ASNR, MSS, DRC) audio will be on the left headphone, and unprocessed audio will be on the right headphone. The processed audio is the channel sent to the IBM Watson servers for transcription.
12. Say the trigger phrase *Hello Blue Genie*. The XF LED on the EVM5517 will illuminate a positive trigger of the keyword.

13. The system records 4.5 s of audio subsequent to a positive recognition of the trigger phrase. As an example, say *Testing 1 2 3*. The Sharp LCD mounted on the CC3220SF-LAUNCHXL will have a message *Recording* (see Figure 45).

Figure 45. Sharp ® LCD Indicates Recording in Progress

14. Once the recording is complete, the Sharp LCD mounted on the CC3220SF-LAUNCHXL has a message *sending* (Figure 46).

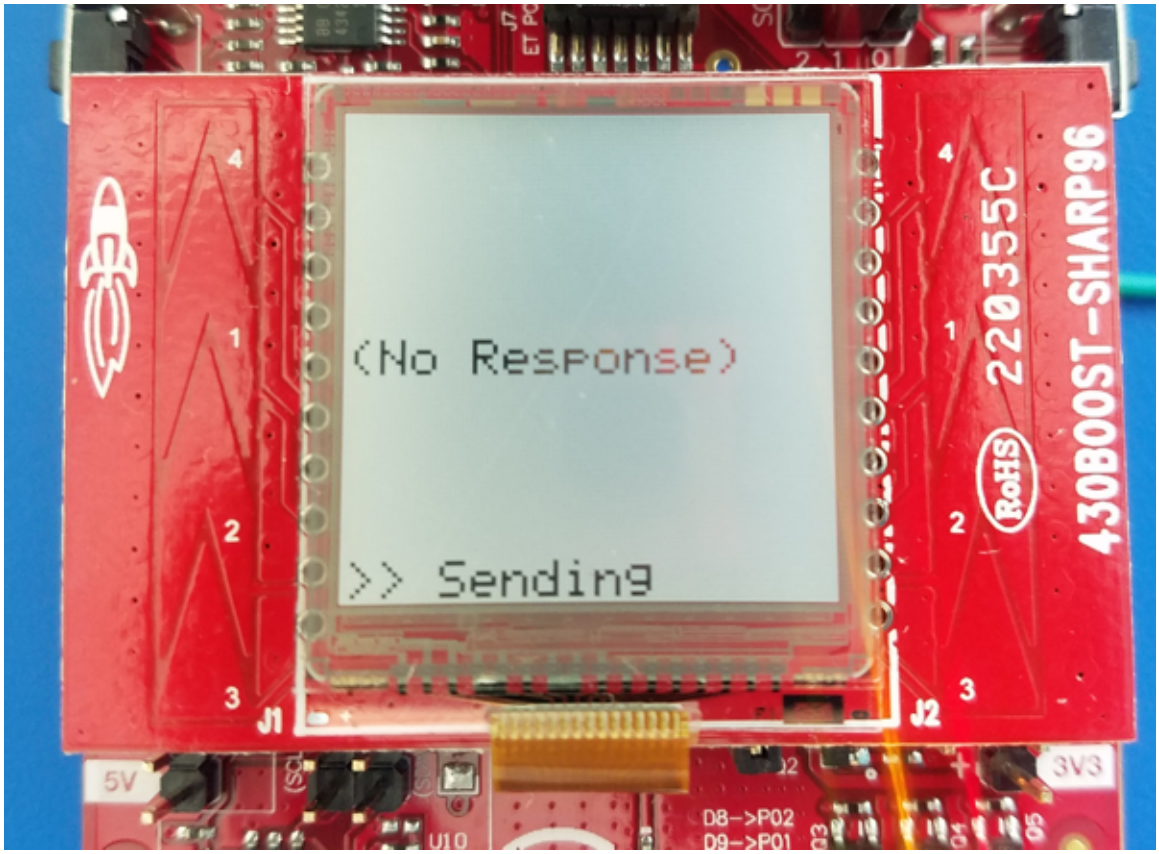


Figure 46. Sharp ® LCD Indicates Sending Audio to IBM Watson ® Servers in Progress

15. After about a 10-s lapse, the transcribed audio display on the Sharp LCD as *Testing 1 2 3*. An accuracy percentage of the transcribed text as reported by the IBM Watson server also displays on the LCD (see Figure 47). If this project is run using CCS (that is, with the debugger connected), the transcription will also be visible on the CCS console.

Figure 47. Sharp ® LCD Shows Transcribed Text

16. The system is now ready for the trigger phrase *Hello Blue Genie* to perform another test. The Sharp LCD mounted on the CC3220SF-LAUNCHXL has a message *recording*.

3.2.1.2 BOOST5545ULP and CC3220SF-LAUNCHXL Demonstration

This section assumes the following:

- The hardware is connected as described in Section 3.1.1.2.1.
- CCS has been installed and launched.
- The set of tools and utilities from Section 2.2.12 have been installed on the PC. The directory structure with the dependencies must look like Figure 48 and Figure 49.

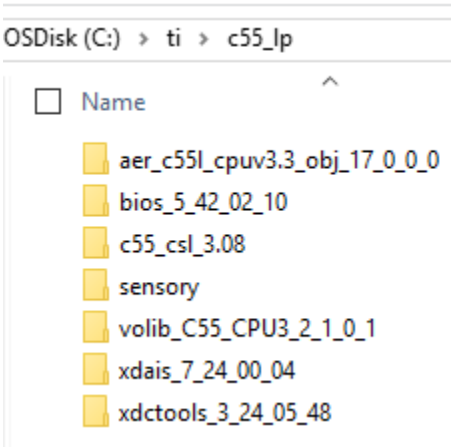


Figure 48. Directory Structure of Project Dependencies and CSL Installation on PC

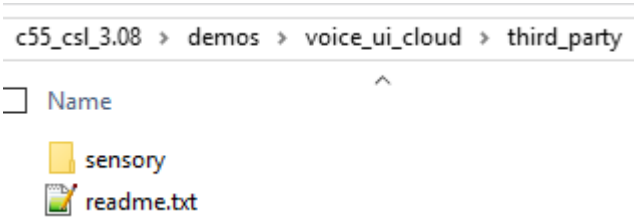


Figure 49. Location of Sensory Dependency Installation

- The C55x CSL has been installed on the PC .
- The Sensory library has been installed at <CSL INSTALL PATH>\c55_lp\c55_csl_3.07\demos\voice_ui_cloud\third_party
- A target configuration for BOOST5545ULP has been created (not covered in this design guide).
- A target configuration for CC3220SF-LAUNCHXL has been created (not covered in this design guide).
- Ensure the CC3220SF-LAUNCHXL is powered and its binary has been flashed as described in Section 3.1.2.3. The CC3220SF-LAUNCHXL should also be connected to a Wi-Fi hotspot with an Internet connection.

NOTE: A pre-built binary that can be loaded and run through CCS is provided in the C55x CSL package at *<CSL ROOT>\demos\voice_ui_cloud\c5545\Debug\c5545_voiceuicloud.out*.

If a bootable SD card image is required, the image is in the same location and is named *bootimg.bin*. For instructions on creating and booting from an SD card on the BOOST5545ULP, see Section 6.

1. With CCS launched, import the C5545 project and C55XXCSL_LP into the workspace from *<CSL INSTALL ROOT>\demos\voice_ui_cloud\c5545* and *<CSL INSTALL ROOT>\ccs_v6.x_examples* respectively. The workspace will look like Figure 50.

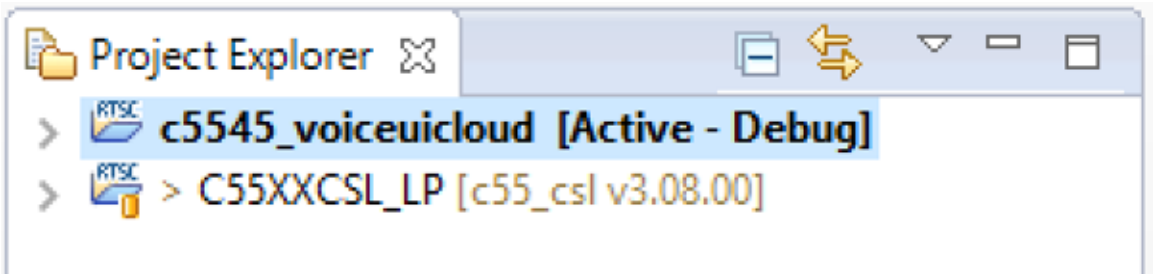


Figure 50. BOOST5545ULP and CC3220SF-LAUNCHXL Demonstration CCS Workspace

2. Before building the demonstration, ensure csl_general in C55XXCSL_LP has the correct macro for the C5545 #define C5545_BSTPCK. The file is located at *<CSL ROOT>\inc*. Once the correct macro is defined, rebuild the C55XXCSL_LP project.
3. Right-click on the c5545_voiceuicloud project, and click build project.
4. When the build completes, c5545_voiceuicloud.out will be present in the *Debug* folder.
5. Connect the BOOST5545ULP and CC3220SF_LAUNCHXL as illustrated in Figure 14. Power on the BOOST5545ULP and connect a USB cable to the onboard debugger J9.
6. Ensure the CC3220SF-LAUNCHXL is powered and its binary flashed as described in Section 3.1.2.3. The CC3220SF-LAUNCHXL must also be connected to a Wi-Fi hotspot with an Internet connection. If desired, the CC3220SF-LAUNCHXL debug information can be viewed in a terminal emulation software such as TeraTerm. See Section 3.1.2.3.2 step 12 for details on connecting to the CC3220SF-LAUNCHXL terminal window.
7. Launch the target configuration for the BOOST5545ULP.
8. Right-click and connect to the DSP core with a GEL file loaded.
9. Load c5545_voiceuicloud.out onto the core.
10. Press resume, and there will be debug information from the initialization of the LMB printed on the CCS console. A blue LED on the LMB will illuminate, which indicates a successful initialization of the onboard PCM1864.

11. At this point the demonstration is ready to run and the word *Ready* will be printed on the BOOST5545ULP OLED (see Figure 51). The audio captured on the LMB will be processed using BF, ASNR, MSS, and DRC.The processed audio will then be sent to the IBM Watson servers for transcription.

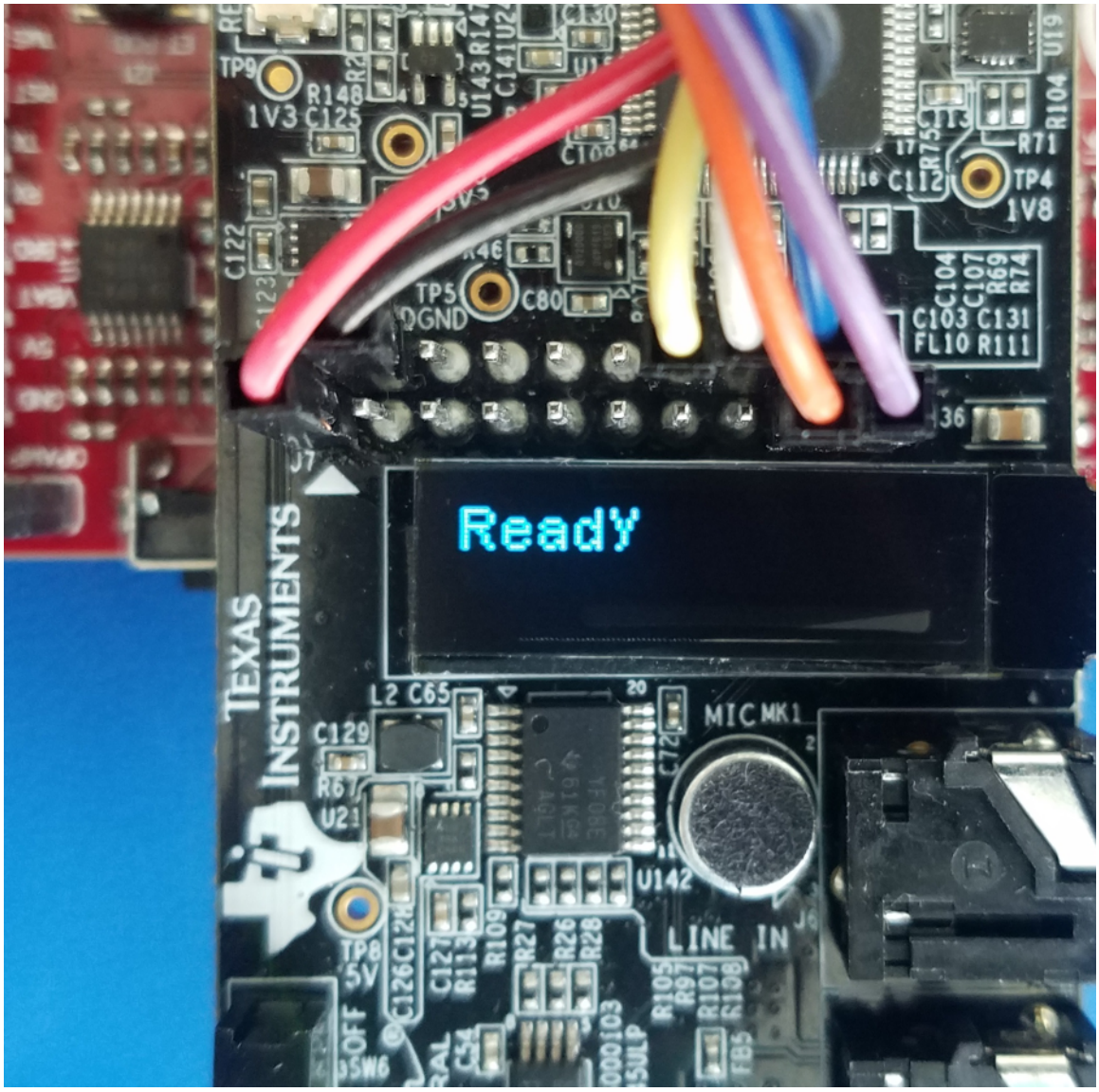


Figure 51. BOOST5545ULP OLED Shows System is Ready for Trigger Phrase

12. Say the trigger phrase *Hello Blue Genie*.The trigger phrase appears on the OLED (see Figure 52).

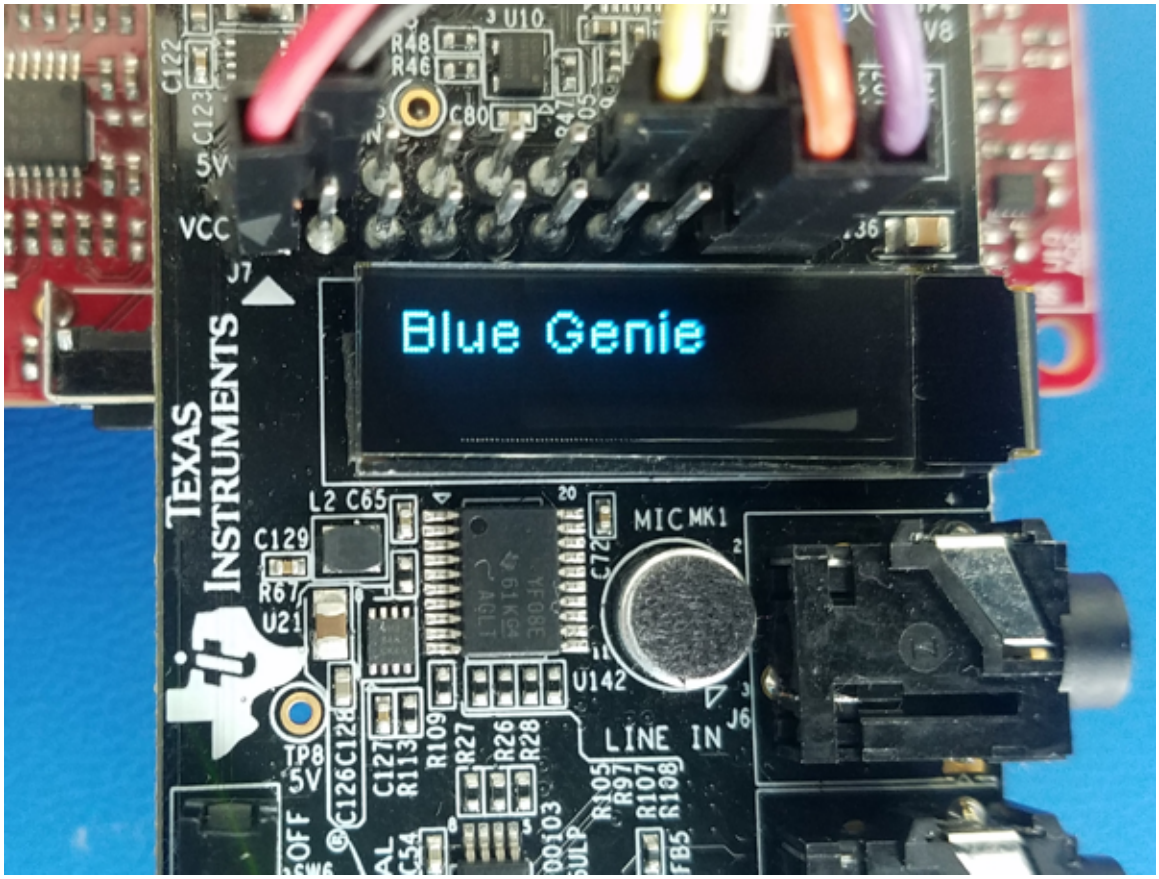


Figure 52. BOOST5545ULP OLED Shows Trigger Phrase After Positive Recognition

13. The system records 4.5 s of audio subsequent to a positive recognition of the trigger phrase. As an example, say *Testing 1 2 3*. Once the 4.5 s have elapsed, the audio clip is sent to the IBM Watson servers for transcription. The roundtrip takes about 10 s varying by network connectivity (see Figure 53).

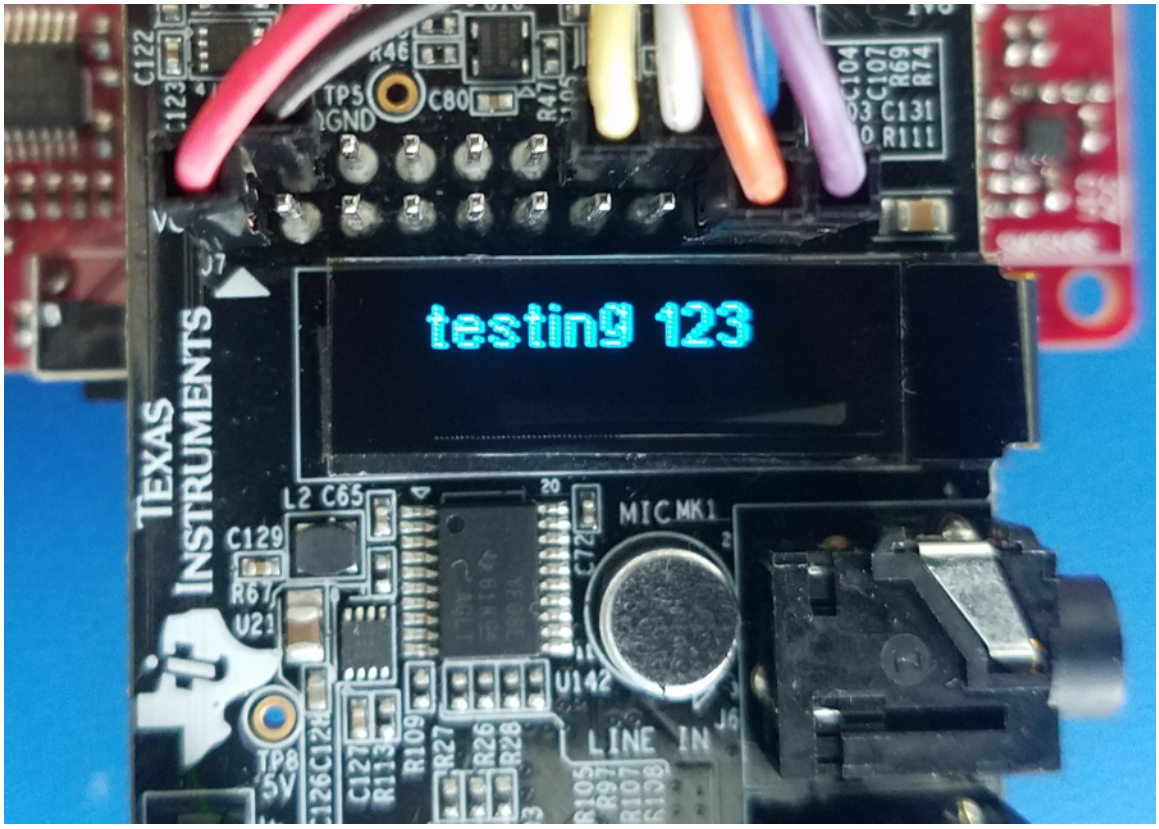


Figure 53. BOOST5545ULP OLED Shows Transcribed Text

14. After about 10 s have elapsed, the transcribed audio is displayed on the BOOST5545ULP OLED as *Testing 1 2 3*.
15. The system is now ready for the trigger phrase *Hello Blue Genie* to perform another test.

3.2.2 Test Results

This section details the test results based on general execution of the demonstrations.

Table 12. Server Response Time

TEST PARAMETER	RESULT	COMMENTS
IBM Watson server roundtrip time	Approximately 10 s	The roundtrip time varies on network conditions and IBM server response times.

Table 13. Performance Measurements (1)

PARAMETER	C5517	C5545
Total MIPS	200 MHz	120 MHz
Number of physical microphones	4 (2 × I2S)	2 (1 × I2S)
Number of virtual microphones	4	2
Measured MIPS (2)	33+27+1+10 = 71 MIPS	10+13+1+10 = 34 MIPS
Memory usage (SARAM+DARAM)	240KB used, 80KB remaining	215KB used, 105KB remaining

(1) C5545 and C5517 have 320-KB total on-chip memory
(2) Measured MIPS includes BF+ASNR+MSS+Sensory

4 Design Files

To download the design files, such as schematics, bill of materials (BOM), PCB layout recommendations, and so on, see the design files at:

- TIDEP-0083
- TIDA-01470
- CC3220SF-LAUNCHXL
- BOOST5545ULP
- EVM5517

5 Software Files

To download the software files, see the design files at TIDEP-0083.

6 Related Documentation

1. Chen, Joe C., Kung Yao, and Ralph E. Hudson, *Acoustic Source Localization and Beamforming: Theory and Practice* , EURASIP Journal on Advances in Signal Processing 2003, no. 4 (2003): 359-70.
2. Wikipedia, Beamforming
3. Texas Instruments, *C5517 General Purpose EVM User Guide*
4. Texas Instruments, C5517 Evaluation Module
5. Texas Instruments, *TMS320C5545 BoosterPack Hardware User's Guide*
6. Texas Instruments, *Creating an SD card boot image for C55x*
7. Texas Instruments, *CC3120 & CC3220*
8. Texas Instruments,C55x Chip Support Libraries (CSL) Download
9. Sensory Inc., Sensory Truly Handsfree keyword recognition library
10. Texas Instruments, Code Composer Studio
11. Texas Instruments, Demonstrating Triggering and Control with Cloud Connection to IBM Watson
12. Texas Instruments, SimpleLink CC3220 SDK

6.1 Trademarks

7 About the Authors

LALINDRA JAYATILLEKE is a digital applications engineer at TI specializing in embedded processing applications including audio processing systems. Lalindra earned his bachelors of science in electrical engineering from the University of the District of Columbia, Washington DC.

MICHAEL REYMOND is an applications engineer at Texas Instruments, where he is responsible for supporting customers designing Wi-Fi-enabled systems. Michael earned his bachelors of science in computer engineering from Georgia Tech.

MING WEI is a senior software engineer at TI, where he develops and supports the Processor SDK RTOS for Sitara and the DSP families of SOC devices. Ming brings his extensive experiences and knowledge in real-time systems, signal processing, and code optimization to this role. Ming earned B.Sc., M.Sc., and Ph.D. in computer sciences from Xi'an Jiao-tong University and University of North Texas.

7.1 Acknowledgments

JOHN GODBEY is a software engineer at TI supporting supports the Processor SDK RTOS for Sitara and the DSP families of SOC devices.

HARI PATEL is a catalog embedded processors Intern at TI, where he is working on developing firmware for C55x DSPs, MSP430™ microcontrollers, and TI Simplelink family devices. Hari is pursuing his master of science in electrical engineering from University of Florida, Gainesville, FL.

JORGE TRONCOSO is a catalog embedded processors Intern at TI. Jorge is an undergraduate studying electrical engineering and computer science at MIT.

Revision A History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (October 2017) to A Revision	Page
• Added row for Sensory TrulyHandsFree (THF)	3
• Added Section 2.2.7	8
• Added row for Sensory THF 1.0.0	9
• Added third footnote to Table 2	9
• Added links in Table 10	22
• Added added third bullet point, Figure 42, and Figure 43	44
• Added added third bullet point, Figure 48, and Figure 49	49
• Changed link in list item 9	54
• Added list item 12	54

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265