TI Designs

Audio Communication With Power Over Ethernet (PoE)

TI Designs

This audio communication design uses Texas Instrument's TM4C129x high performance microcontrollers (MCUs) with integrated Ethernet physical interface device (PHY) to capture, exchange, and play back audio efficiently. The Power over Ethernet (PoE) solution supplies power over the network instead of over separate external power sources. The application uses TivaWare™ Software to implement a user interface with touch control, lwIP to exchange of data over the network, and Opus audio codec to compress audio data to improve bandwidth usage.

Design Resources

TIDM-TM4C129POEAUDIO	Design Folder
TM4C129ENCPDT	Product Folder
TPS23753A	Product Folder
TPD2E2U06	Product Folder
TLV431A	Product Folder
TPS73733-Q1	Product Folder
OPA322	Product Folder
LM4819	Product Folder
TIDM-TM4C129POE	Tools Folder
BOOSTXL-K350QVG-S1	Tools Folder

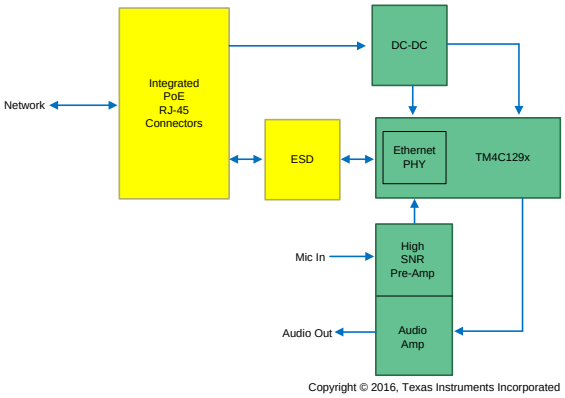
TI E2E™ Community — ASK Our E2E Experts

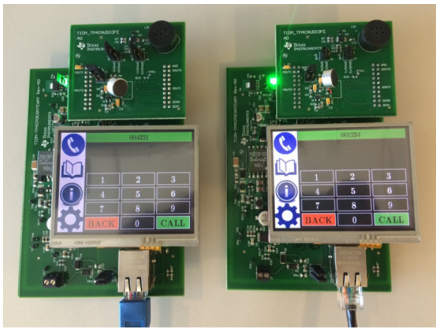
Design Features

- Less Than 500 mW of Power Consumption for the MCU, LCD, and Audio Subsystem With Total System Provision of up to 7-W Isolated Output From the Flyback Converter
- Efficient and Royalty-Free, Open-Source Opus Audio Codec to Reduce Network Usage During Audio Call and Software Cost
- Smart Touch LCD Panel Running TivaWare™ Software for a Graphics-Rich User Interface and Experience
- High Signal-To-Noise Ratio (SNR) Microphone Stage to Improve Audio Capture Using Instrumentation Analog-to-Digital Converter (ADC)
- PC Application Software Source Code
- Executable for Dynamic Node Configuration Over the Network

Featured Applications

- Access Control Panels
- Audio Subsystem for Cameras
- Home Security Solutions
- Point-to-Point Communications and In-Home Voice Over IP (VoIP) Audio Monitoring





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

1 Key System Specifications

Table 1. Key System Specifications

PARAMETER	SPECIFICATIONS
Audio sampling rate	48 kHz
Audio bits per sample	12 bits
Number of channels	1 (mono)
Opus encoder bit rate	96 kHz
Opus encoder bit rate type	Constant bit rate (CBR)
Opus encoder frame size	20 ms
Opus encoder complexity	0
Audio playback rate	48 kHz
Power consumption	500 mW (overall)
User input interface	Resistive touch panel
Microphone sensitivity	–47 db ± 5 db
Microphone standard operation voltage	1.5 V
Microphone current consumption	0.5 mA
Microphone impedence	0.68 kΩ
Signal-to-noise ratio (SNR)	> 55 db

2 System Description

The TM4C129x family of MCUs features an integrated Ethernet PHY and MAC with cryptographic modules and many serial interfaces for control and sensor data gain. The TM4C129x adds a layer of intelligence at the remote node when integrated with PoE design. This addition allows customers to leverage their existing network to communicate and control devices securely. The PoE solution delivers power that reduces the system cost in IoT space, which adds value to the products.

2.1 TM4C129ENCDT

The TM4C129ENCPDT is a 120-MHz, high-performance MCU with 1MB of on-chip flash and 256KB of on-chip SRAM. The MCU features an integrated Ethernet MAC+PHY for connected applications and cryptographic modules of advanced encryption standard (AES), data encryption standard (DES), and secure hash algorithm (SHA) for encryption, decryption and authentication. The device has high bandwidth interfaces like a memory controller and a high-speed USB2.0 digital interface. With the integration of many serial communication peripherals, a 12-bit ADC capable of up to 4 MSPS, and motion control peripherals, the device provides a unique solution for a variety of applications ranging from industrial communication equipment to smart energy and smart grid applications. Figure 1 shows the high-level block diagram for this MCU.

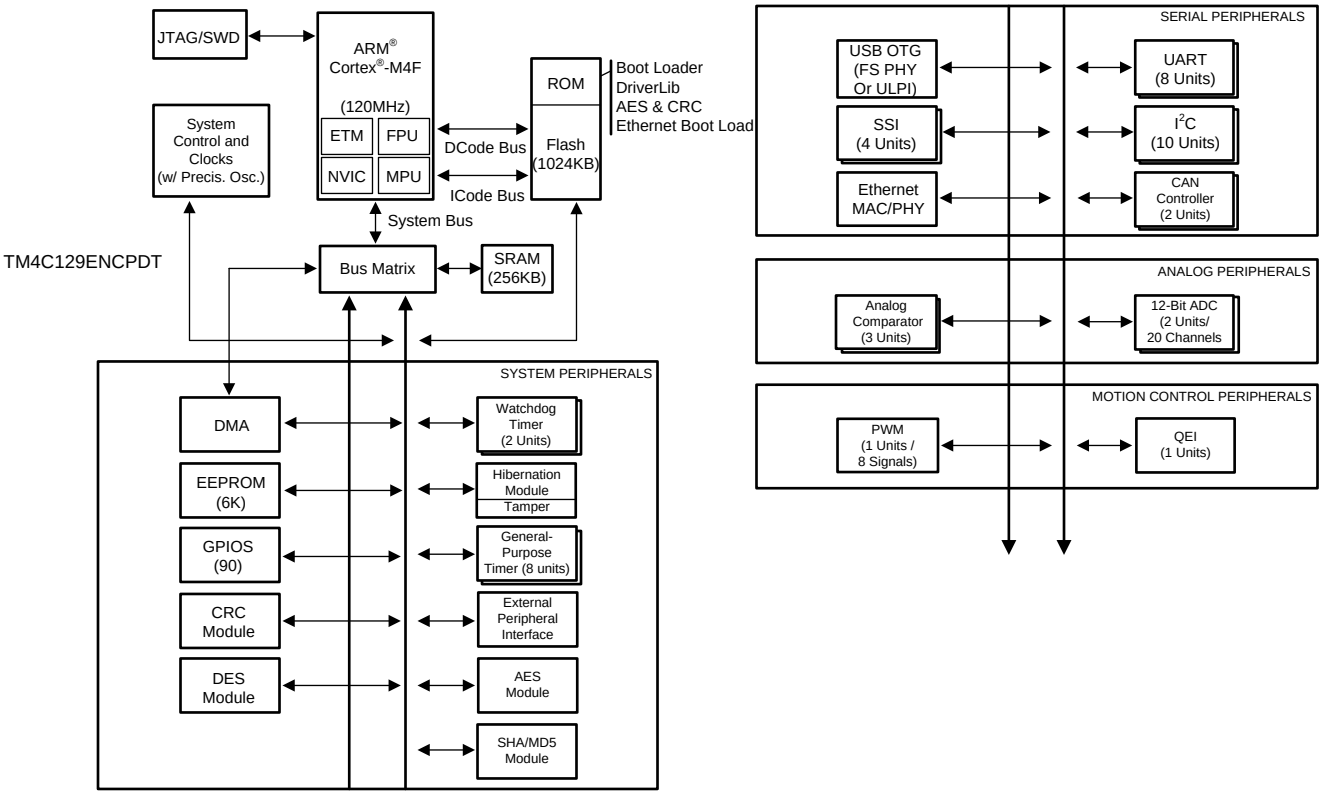


Figure 1. TM4C129ENCPDT MCU High-Level Block Diagram

2.2 TPS23753A

The TPS23753A is a combined PoE powered device (PD) interface and current-mode, DC-DC controller specifically developed for isolated converter designs. The PoE implementation supports the IEEE 802.3at standard as a 13-W, type-1 PD. The requirements these PDs are a superset of IEEE 802.3-2008 (originally IEEE 802.3af). The DC-DC controller features a bootstrap start-up mechanism with an internal, switched current source. This design provides the advantages of cycling-overload fault protection without the constant power loss of a pullup resistor.

2.3 OPA322

The OPA322 series consists of single-, dual-, and quad-channel complementary metal oxide semiconductor (CMOS) operational amplifiers featuring low-noise and rail-to-rail inputs and outputs optimized for low-power, single-supply applications. With a wide supply range of 1.8 to 5.5 V, the low-quiescent current of only 1.5 mA per channel makes these devices well-suited for power-sensitive applications. The OPAx322S models include a shutdown mode that allows the amplifiers to switch from normal operation to a standby current that is typically less than 0.1 µA.

2.4 LM4819

The LM4819 is a mono-bridged power amplifier capable of delivering 350-mW_{RMS} output power into a 16-Ω load or 300-mW_{RMS} output power into an 8-Ω load with 10% THD+N from a 5-V power supply. The LM4819 Boomer audio power amplifier is specifically designed to provide high-quality output power and minimize printed circuit board (PCB) area with surface mount packaging and a minimal amount of external components. Because the LM4819 amplifier does not require output-coupling capacitors, bootstrap capacitors, or snubber networks, it is optimally suited for low-power, portable applications. Figure 2 shows a block diagram of audio communication with the PoE solution.

3 Block Diagram

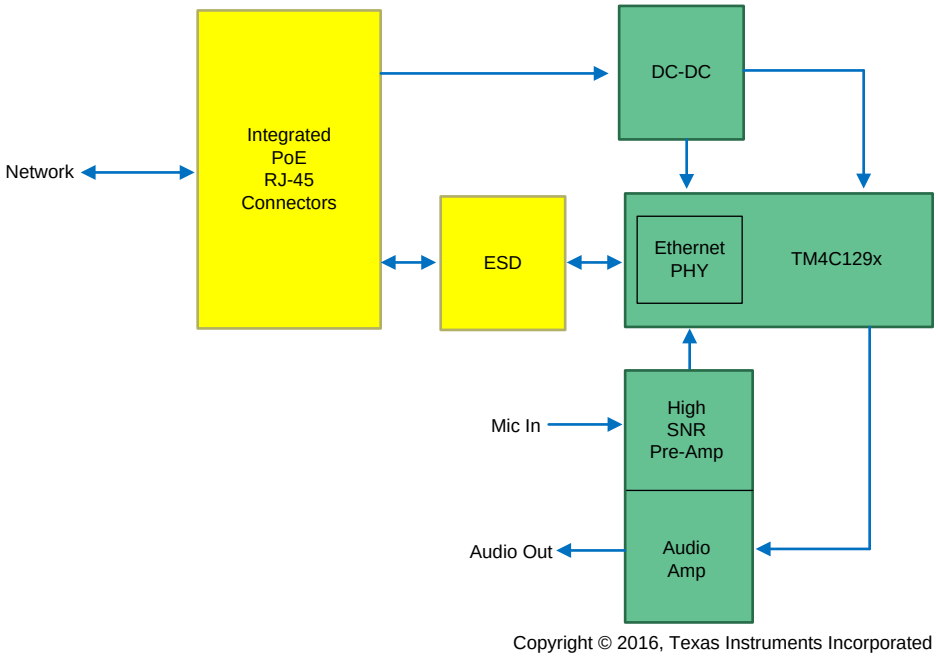


Figure 2. Audio Communication With PoE Block Diagram

4 System Design Theory

This section details the individual software components developed for the audio communication solution. Figure 3 shows the solution block diagram. In this application, each board refers to a node.

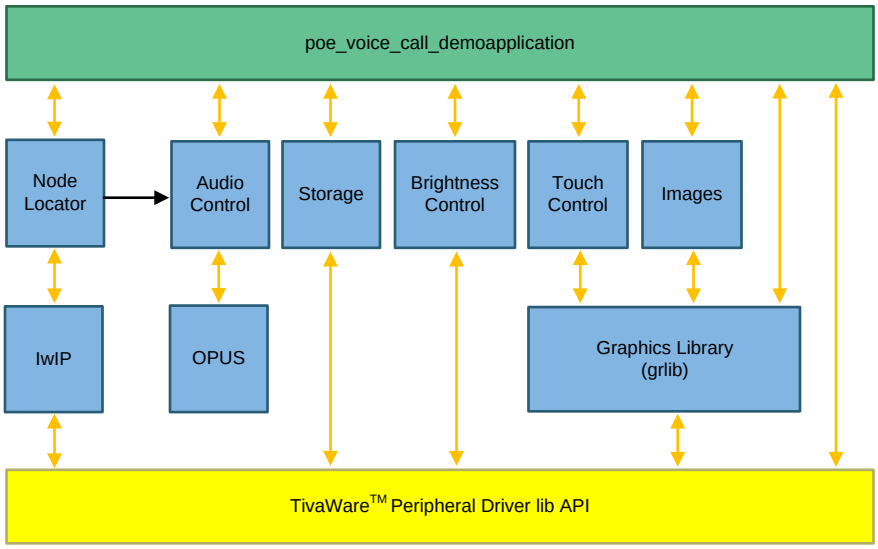


Figure 3. TIDM-TM4C129-POE-AUDIO Software Block Diagram

4.1 Application Layer

The poe_voice_call_demo application integrates the different blocks and runs the graphics for the user interface (UI). The application calls functions from the different blocks based on the inputs provided to the UI functions. The UI splits into windows and panels, each with a specific function for the application.

4.1.1 Layout of the UI

Figure 4 shows the overall layout of the UI consists of the three components.

1. The node number display is located on the top of the LCD screen.
2. The navigation panel is located on the left of the LCD screen.
3. The user window is the rest of the LCD screen.

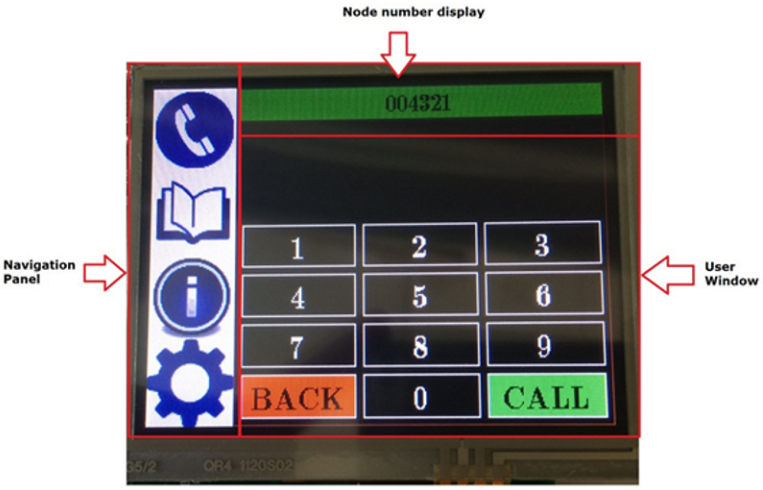


Figure 4. UI Layout

4.1.2 Node Number Display

The node number display shows the calling number for the local node and the node's status. When the node number display is set with a seven-digit numeric value and a green background, the display informs the user that the node is configured. When the node number display shows *INFO UNAVAILABLE* and a red background, the display informs the user that the node is not configured. Figure 7 shows how a touch of the node number display changes the user window layout to the information window.

4.1.3 Navigation Panel

The navigation panel controls the different user windows for the application. Figure 5 shows the four icons on the navigation panel.

1. The call icon brings up the call window, which allows the user to call another node.
2. The call history icon brings up the call history window, which allows the user to see the node number with which audio was exchanged, the duration of the call, the number of bytes received, and the type of call.
3. The information icon brings up the information window, which gives the user information on the local node.
4. The settings icon brings up the setting window, which can change the screen brightness, ringer volume, and LED flash on-call setting. The setting icon can also mute the ringer and clear the call history.

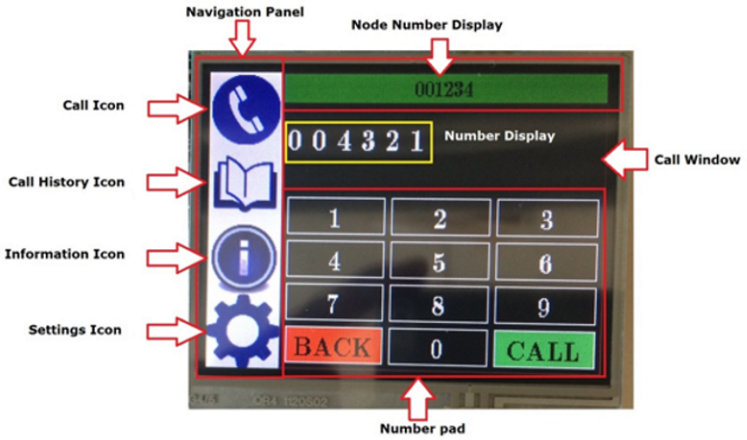


Figure 5. Navigation Panel Layout and Call Window

4.1.4 Call Window

The call window displays at reset or by pressing the call icon on the navigation panel. The call window has two distinct parts as shown in Figure 5:

1. The number pad is a zero through nine numeric pad used to call a remote node. The number pad includes a *BACK* button that erases the last number entered on the node number display and a *CALL* button to place a call to the number entered on the node number display.
2. The number display shows the six-digit number to place the call.

4.1.5 Call History Window

The call history window displays when pressing the call history icon on the navigation panel. The call history window has four distinct parts as shown in Figure 6:

1. The call type icon shows the type of call. A left-pointing, blue arrow signifies an incoming call. A red X signifies a missed or cancelled call. A right-pointing, green arrow signifies an outgoing call.
2. The remote node number shows the node number that performed the audio data exchange.
3. The call duration gives the elapsed call time in hours:minutes:seconds (HH:MM:SS) format. Missed calls will always be 00:00:00.
4. The byte information field gives the number of bytes compressed by the opus encoder on the remote node, which the local node received.

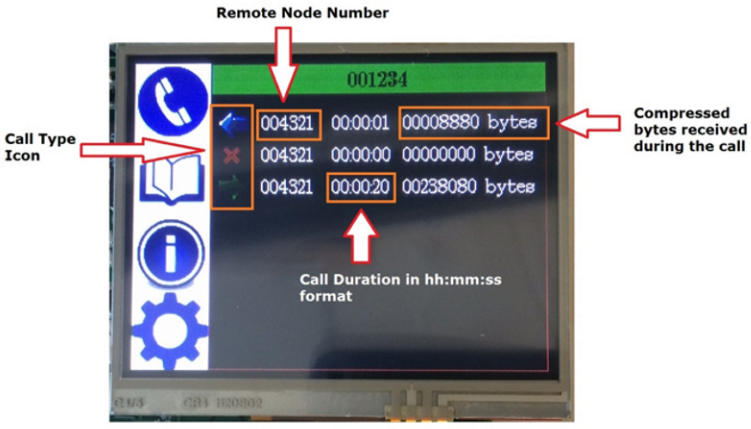


Figure 6. Call History Window

The call history window holds the information for the last eight calls made, received, or missed.

4.1.6 Information Window

The information window displays by pressing the information icon on the navigation panel or by pressing the node number display. The information window provides the four pieces of information as shown in Figure 7:

1. IP address of the local node
2. MAC address of the local node
3. Node number of the local node
4. Software version for the application

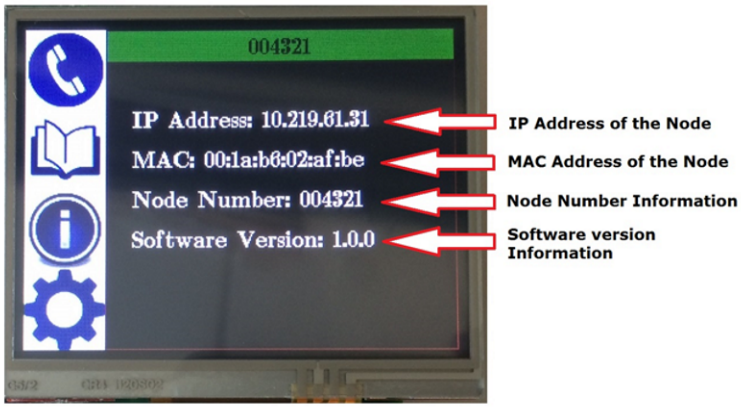


Figure 7. Information Window

4.1.7 Setting Window

The setting window displays by pressing the setting icon on the navigation panel. As shown in Figure 8, this window provides the user with the following controls:

- The *Brightness* control slider moves left to reduce the backlight or moves right to increase the backlight of the panel.
- The *Ringer Volume* control slider moves left to reduce the ringer volume or moves right to increase the ringer volume on a call.
- An onboard LED can indicate a placed or received call. This notification enables by pressing the *LED Flash On Call* button on the setup window. A green check mark indicates it is enabled, and a red X indicates it is disabled.
- The *Ringer On Call* button mutes the ringer during a placed or received call. A green check mark indicates the ringer volume control is in effect. A red X indicates the ringer is muted.
- The *Clear Call History* button erases the call history stored on the device.
- The *Save Setting* button submits the changes made in the setting window to on-chip, nonvolatile memory. The information can be retrieved and used by the application on a reset or power-on.

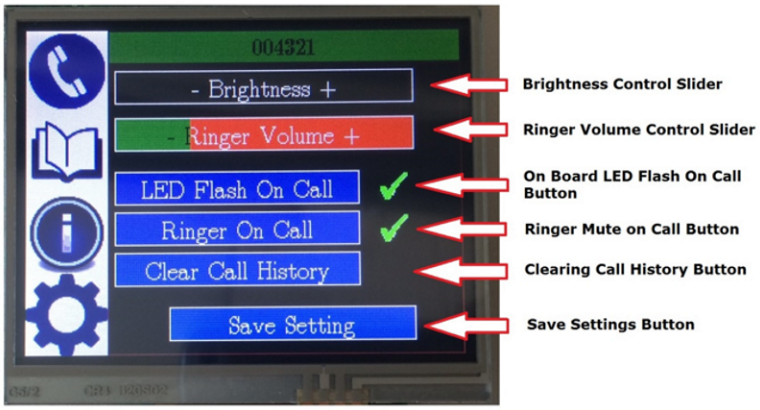


Figure 8. Setting Window

4.2 Node Locator Block

The node locator block, as shown in Figure 3, runs the control-state machine over the network stack (using the lwIP). The device uses the raw user datagram protocol (UDP) application program interface (API), provided by lwIP, to exchange both control data and voice data over the network. In order to provide non-blocking ports on the network, the node locator block uses two unassigned UDP ports.

1. UDP port 64040 for control-data exchange
2. UDP port 64041 for audio-data exchange

4.2.1 Control Data Service Packet

The unassigned UDP port number 64040 provides the control-data services. The API NodeLocatorInit initializes the control-state machine used by the application to place and receive calls and configure an uninitialized node.

Even though UDP exchanges control packets, a request-response mechanism builds so the local and remote node can exchange control packets reliably and detect a loss of either of the packets. Figure 9 shows how both control packets (request and response) have the same 9-byte structure. Byte-0 is always the command; however, there are two types of packets based on the information transmitted.

1. Type-1 control packet has bytes one to eight with custom information.
2. Type-2 control packet has bytes one to four with the node number information of the source node. Type-2 also has bytes five to eight with the node number information of the destination node.

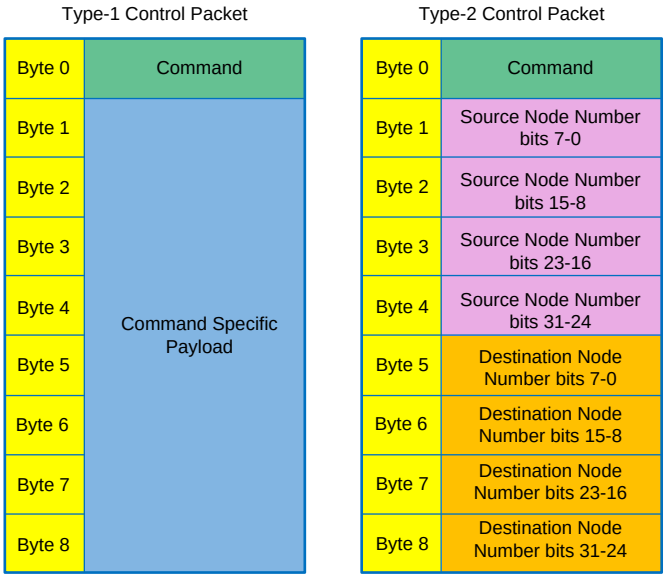


Figure 9. Control Packet Structure

The following list details the commands from Figure 9:

- Command NODE_CFG_NODENO (Command value 0xFF) is a Type-1 packet used by the NodeAdmin.exe utility to configure an uninitialized node with the node number (see Section 7.4).After the node receives the command and configures the node number, it initiates a system reset. This reset allows the application to configure the node number for audio-data exchange.
- Command NODE_CFG_REQINFO (Command value 0xFE) is a Type-2 packet used by the NodeAdmin.exe utility to request the node number information from the nodes available on the network. This command uses a broadcast IP address.
- Command NODE_CFG_RESPINFO (Command value 0xFD) is a Type-2 packet used by the nodes on the network to respond to the NodeAdmin.exe. The command contains the currently configured node number.
- Command NODE_FIND_REQ (Command value 0xEF) is a Type-2 packet issued by the calling node to find a remote node by its node number. This command uses a broadcast IP address.
- Command NODE_FIND_RESP (Command value 0xEE) is a Type-2 packet issued by the remote node in response to the NODE_FIND_REQ issued by another node on the network.
- Command NODE_CALL_REQ (Command value 0xED) is a Type-2 packet issued by the calling node to place a call to the remote node.
- Command NODE_CALL_RESP (Command value 0xEC) is a Type-2 packet issued by the remote node to the calling node in response to a NODE_CALL_REQ. Once this packet is issued, the two nodes can exchange audio data.
- Command NODE_CALL_BUSY (Command value 0xEB) is a Type-2 packet issued by the remote node to the calling node in response to a NODE_CALL_REQ if it is already in an active call with another node. This helps the calling node terminate the call automatically.
- Command NODE_CALL_TERM (Command value 0xEA) is a Type-2 packet issued by either the local or remote node if an active call has to be terminated or in response to a NODE_CALL_REQ.

4.2.2 Control-Data, Service-State Machine

The control-data services use the packets in Section 4.2.1 to exchange information on the state of the node. The node locator block uses these packets to run a control-state machine. The application uses the control-state machine to control other blocks, like audio and graphics. The control-state machine consists of six states.

1. The NODE_RESET state enters if the application on the local node is unable to locate a node number

in the nonvolatile memory.

2. The NODE_CONFIG state enters when the PC application requests an idle node to reprogram certain information. The current implementation reserves this state.
3. The NODE_IDLE state enters when the local node is configured with a valid node number and is not in an active call state.
4. The NODE_FIND state enters when the local node attempts to find a remote node.
5. The NODE_WAITCALL state enters when the local node finds a remote node, places a call request, and awaits a response.
6. The NODE_INCALL state enters when the local node is in an active call.

The flowcharts in Figure 10, Figure 11, Figure 12, and Figure 13 describe how the node-locator block handles the control-state machine transition, which depends on the packets received and their response to the remote node or PC.

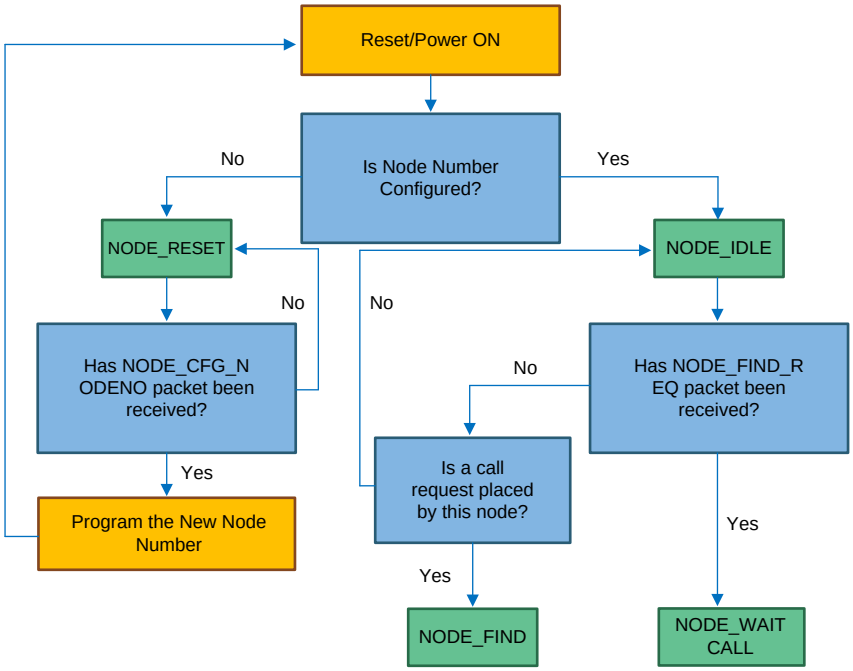


Figure 10. Control-State Machine One

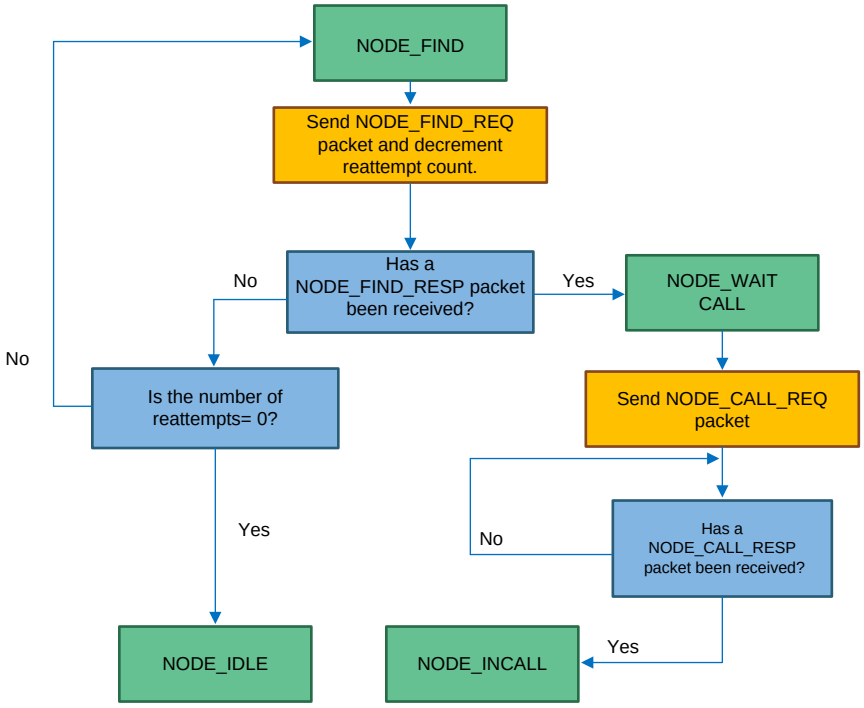


Figure 11. Control-State Machine Two

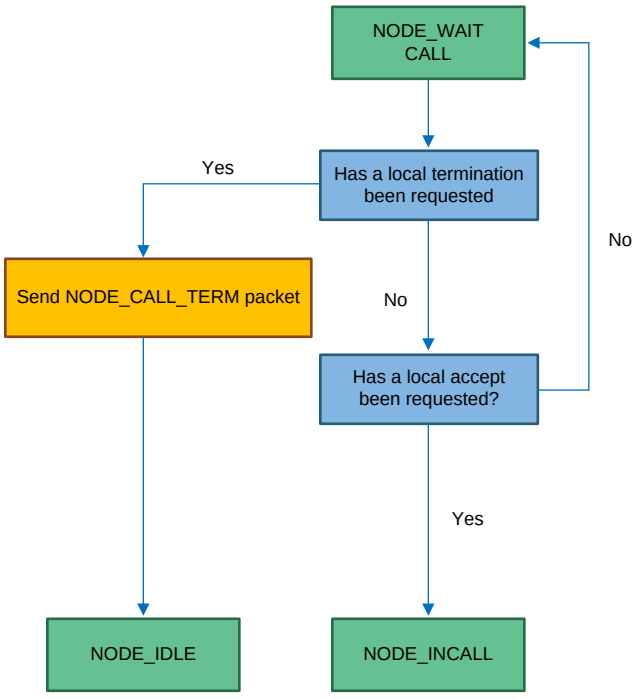


Figure 12. Control-State Machine Three

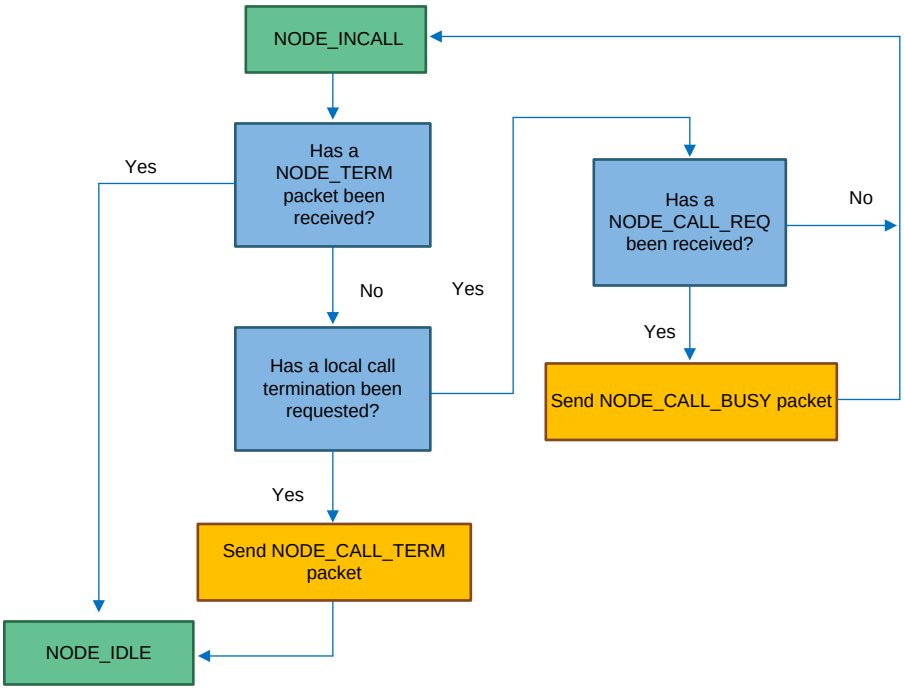


Figure 13. Control-State Machine Four

4.2.3 Audio Data Services

The unassigned UDP port number 64041 provides the audio-data services. The application uses the API NodeLocatorSendData through the audio-control block to send a packet of compressed data over the network to the remote node. Similarly the API NodeLocatorData calls the callback function of the application through the audio-control block to receive a packet of compressed data over the network from the remote node. Both APIs only have effect when the control-state machine indicates to the application that a call is in progress.

4.3 Audio-Control Block

As shown in Figure 3, the audio-control block is responsible for initializing the audio interfaces, capturing the voice from the microphone, compressing the raw audio data, and decompressing the received audio data. The block is also responsible for the playback of the decompressed raw data.

4.3.1 Initializing the Audio Interface

The application calls the API AudioInit to initialize the audio interfaces and data paths. The audio interface initialization requires the system clock frequency. This frequency generates the 48kHz of sampling and playback rate.

4.3.2 Voice Capture and Compression Path

The ADC one, sample sequencer two, and ADC channel nine create the voice capture. The ADC zero uses the external pin trigger since the timer trigger is already used in the touch-control block. This pin triggers by timer 3B with a configuration of a frequency of 48 kHz in pulse width modulation (PWM) mode. The pin generates a rising edge trigger to the ADC for sampling.

A sample and hold of 64 clocks configures the ADC to ensure that the internal capacitor is well-charged during the sampling period before the conversion starts. The direct memory access (DMA) channel's configuration is for the ping-pong mode to transfer 20 ms worth of ADC samples. When either the ping or pong buffer completes, the opus encoder calls to compress the data.

4.3.3 Voice Decompression and Playback Path

The voice playback uses the Timer 0B in PWM mode. The timer uses a PWM period corresponding to 48 kHz in 16-bit mode.

The DMA channel enables for the timer in ping-pong mode. When the network receives a compressed data packet, it first runs the opus decoder to get the raw PCM data. This data copies to either the ping or pong buffer based on which is available. The DMA's primary or alternate control structure enables so that, on the next DMA request, the data transfers to the timer's match register. Based on the match register value, the timer generates a corresponding PWM duty cycle, which is then passed through an external filter to reconstruct the audio.

4.3.4 Volume Control

The audio block also controls both ringer volume and in-call volume control. The audio block uses the shift operation on the audio sample to achieve these controls. If the volume control from the application requires the volume be higher, the audio block performs a left-shift operation. Similarly, if the volume control from the application requires the volume be lower, the audio block performs a right-shift operation on the playback audio data.

4.4 Touch-Control Block

The application uses the touch-control block (see Figure 3) to detect a press on the screen, to retrieve the X and Y coordinates of the press, and to call the necessary callback function to process the press on the LCD panel. The touch-control block uses the ADC 0, ADC 10, and ADC 11 channels. The ADC is triggered by timer 1B every 2 ms to sample the data on the ADC channel and to compute X and Y coordinates of a press.

4.5 Brightness-Control Block

The brightness-control block (see Figure 3) controls the amount of backlight current the application provides to the LCD panel. Generator two uses a PWM output of two kHz to control the brightness of PWM zero. The brightness slider on the setting window allows access to the block. There are three APIs available to the main application.

1. The BrightnessInit API initializes the PWM generator with the two kHz PWM output and a duty cycle as configured by the application.
2. The BrightnessSet API changes the duty cycle during active operation.
3. The BrightnessGet API retrieves the duty cycle during active operation.

4.6 Storage Block

The application uses the storage block (see Figure 3) to save and retrieve call history and settings of the current node. The application uses the EEPROM as the nonvolatile location to reduce the main-flash usage. The main application accesses the block during a call, a touch of the call history icon, a touch of the setting icon, and a touch of the *Save Setting* push button in the settings window. The following APIs are available to the main application.

- The StorageInit API enables the clock to EEPROM and initializes the EEPROM controller.
- The SaveUserSettings API saves the brightness, LED flash on-call, ringer volume, and ringer on-call from the settings window.
- The GetUserSettings API retrieves the brightness, LED flash on-call, ringer volume, and ringer on-call when pressing the settings icon.
- The SaveCallHistory API stores the call type, remote node number, call duration, and number of bytes received by the local node when the local or remote node terminates a call.
- The GetCallHistory API retrieves the call type, remote node number, call duration, and number of bytes received by the local node when pressing the call history icon.
- The GetNumofCallEntry API retrieves the number of call entries in the EEPROM and the current index of the pointer that maintains the history.
- The ClearCallHistory API erases the EEPROM when pressing the clear call history button in the setting

window.

Table 2 shows the layout of the EEPROM that the storage block uses to interact with the main application.

Table 2. EEPROM Layout

STORAGE ELEMENT NAME	EEPROM ADDRESS	DESCRIPTION
STORAGE_SETTING_VALID	0x000	When value is 0x1 this indicates the settings are valid. At the first power up, it will have the value 0xFFFFFFFF. This value indicates that the default settings must be applied.
STORAGE_SETTING_BRIGHT	0x004	This element stores the duty cycle of the PWM for brightness control.
STORAGE_SETTING_SCRFLSH	0x008	This element stores the LED flash on-call setting.
STORAGE_SETTING_RING	0x00C	This element stores the ringer enabled or mute setting.
STORAGE_SETTING_VOLUME	0x010	This element stores the ringer volume level on-call setting.
RESERVED	0x010-0x03C	RESERVED
STORAGE_CALL_POINTER	0x040	This element contains the index value to the most recent call.
RESERVED	0x044-0x04c	RESERVED
STORAGE_CALL_ENTRY_TYPE	0x050 + N × 0x010	This element stores the call type. For each of the 8 entries, this field is increments 0x010.
STORAGE_CALL_ENTRY_NUM	0x054 + N × 0x010	This element stores the duration of the call in seconds. For each of the 8 entries, this field is incremented by 0x010.
STORAGE_CALL_ENTRY_DUR	0x058 + N × 0x010	This element stores the duration of the call in seconds. For each of the 8 entries, this field is incremented by 0x010.
STORAGE_CALL_ENTRY_BYTE	0x05C + N × 0x010	This element stores the number of bytes received. For each of the 8 entries, this field is incremented by 0x010.

4.7 Images Block

The images block (see Figure 3) contains the preprocessed images in PNM format for navigation panel icons, check marks, call type icons, and the call accept or reject buttons. To update any of the existing images, the user must convert the image to PNM format and then use the TivaWare tool *pnmtoc.exe* to convert the image to a C structure, which can then be compiled, linked, and used by the application.

Table 3 gives the size and format of each of the images stored in the image block.

Table 3. Image Properties

IMAGE NAME	CONSTANT NAME	IMAGE FORMAT	HEIGHT	WIDTH
Call decline icon	g_pui8Decline	8BPP	50	50
Call accept icon	g_pui8Accept	4BPP	50	50
Green check mark	g_pui8GreenLight	4BPP	24	24
Red X mark	g_pui8RedLight	8BPP	24	24
Outgoing call icon	g_pui8OutgoingCall	8BPP	24	24
Incoming call icon	g_pui8IncomingCall	4BPP	24	24
Missed call icon	g_pu8MissCall	8BPP	24	24
Information icon	g_info	8BPP	60	60
Call history icon	g_dir	8BPP	60	60
Call menu icon	g_call	8BPP	60	60
Setting menu icon	g_setting	8BPP	60	60
Information icon on press	g_iinfo	8BPP	60	60
Call history icon on press	g_idir	8BPP	60	60
Call menu icon on press	g_icall	8BPP	60	60
Setting menu icon on press	g_isetting	8BPP	60	60
Splash screen on startup	g_pui8Splash	8BPP	240	320
Call stop or decline button	g_CallStop	8BPP	32	32
Call accept button	g_CallStart	8BPP	32	32

5 Getting Started Hardware

To create the solution, TIDM-TM4C129POE must be used as the base board to which the LCD BoosterPack™ and the audio board can be connected. As the solution is PoE, the user must have a PoE switch available that powers the board during configuration, firmware download, and to run the application.

5.1 TIDM-TM4C129POE Base Board

The TIDM-TM4C129POE board has the power stage and the TM4C129ENCPDT MCU. Figure 14 shows the shunts that must be placed for the solution. Figure 14 also shows the connectors and headers that the audio and display board use.

The user must make sure that the 3.3-V, 5.0-V, and Ethernet power shunts are mounted (see the yellow boxes in Figure 14).

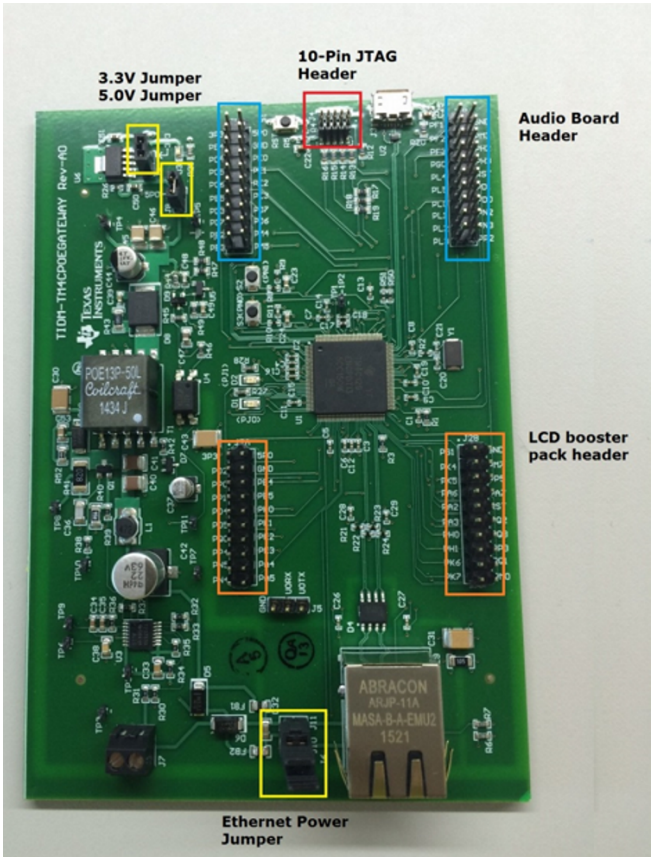


Figure 14. TIDM-TM4C129POE Connector and Shunt Layout

5.2 Audio Board Interfacing

Figure 15 shows how the audio board must mount on the upper pair of headers of the base board. The audio board primarily has the microphone and a high-SNR, preamplifier stage to capture audio input. The board also has an audio amplifier and a speaker for playing audio. There is a microSD card slot on the bottom of the audio board, which is optional to save audio data, playback, preconfigured audio files, and so on.

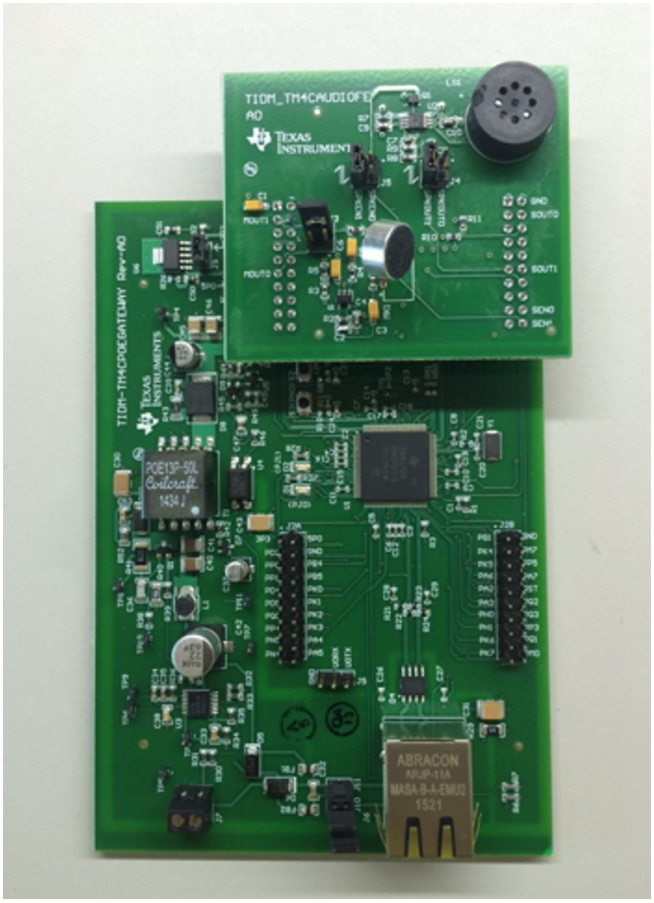


Figure 15. Mounting Audio Board

5.3 LCD BoosterPack™ Interfacing

The solution uses the BOOSTXL-K350QVG-S1 BoosterPack which must be mounted on the lower pair of headers as shown in Figure 16. The LCD panel provides both the display and touch interface for the firmware that the solution executes.

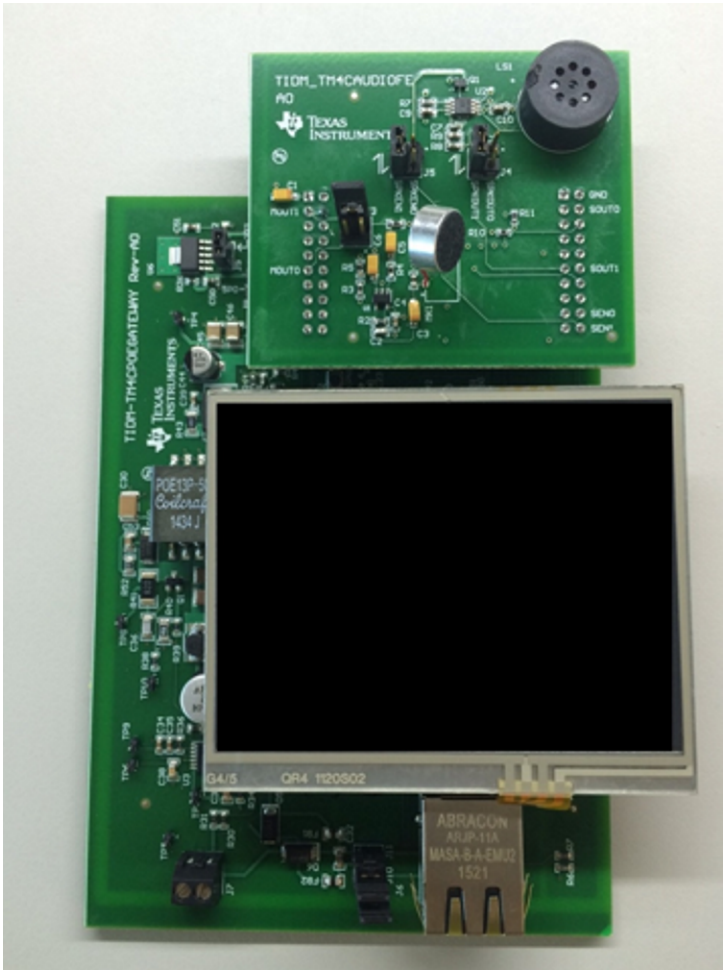


Figure 16. Mounting the LCD Touch Panel

5.4 Connecting the Debugger

There is a 10-pin, 50 mil ARM® JTAG header on the board. This device configures the MAC address for the TM4C129x MCU and downloads the application firmware. Because the audio board extends from the top of the header, the user must either remove the audio board when programming the MCU for the first time or place a corresponding flat ribbon cable (FRC) before mounting the audio board as shown in Figure 17.

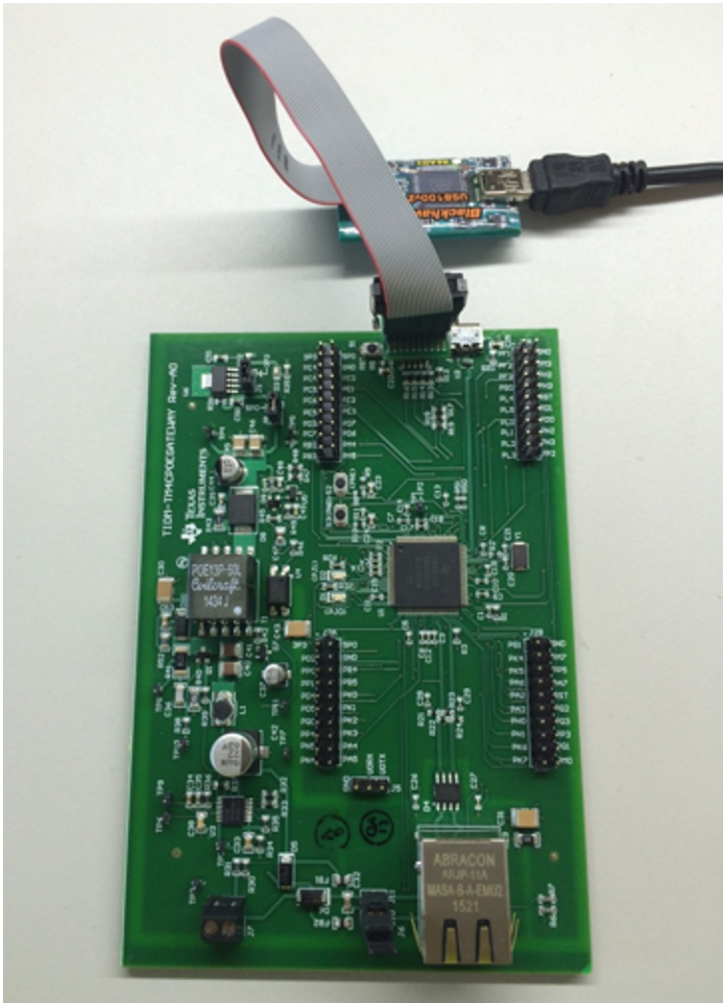


Figure 17. Connecting the Debugger

5.5 Powering the Solution

To power the solution, the user must connect the base board to a PoE-enabled Ethernet switch. This TI Design shows the NetGear GS110TP, 8-port PoE switch. When the Ethernet cable connects between the switch and the board, the green power-on LED lights (see Figure 18). The board is now ready to program the MAC address and application firmware.

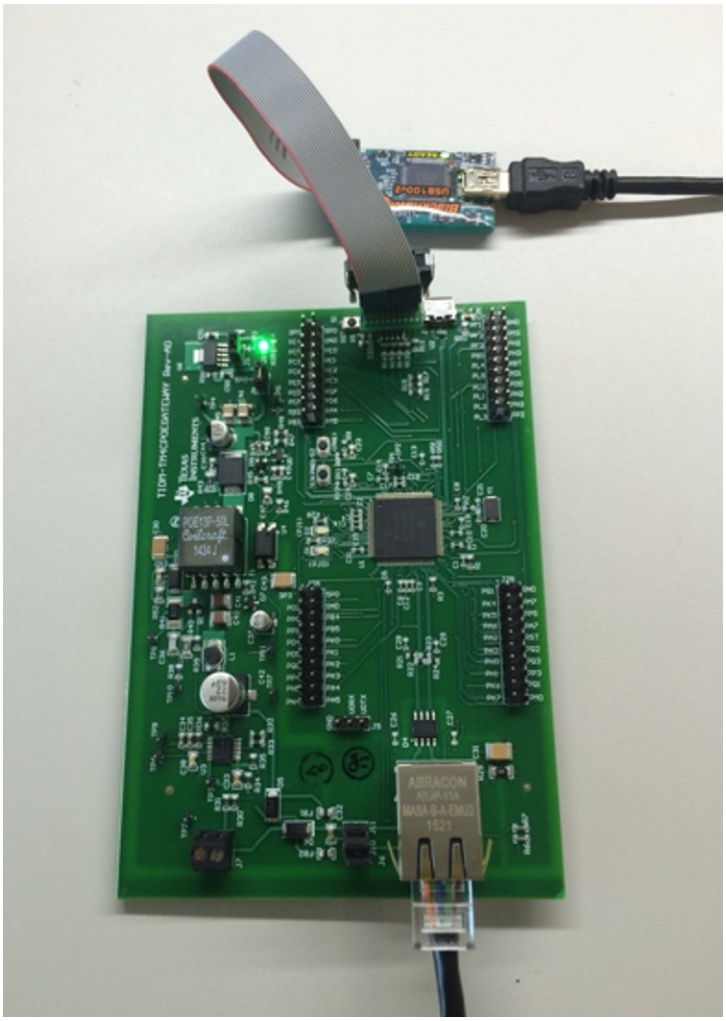


Figure 18. Powering the Solution

6 Getting Started Firmware

The software for this reference design comes as an installer that the user must install on a PC. It normally takes about a minute for the installer to execute. The following tools are required for rebuilding the project firmware:

- TivaWare 2.1.3
- The project collaterals for SPMA076
- Opus source code, version 1.1.2
- CCS 6.1.1.00022 with ARM compiler tool chain, version 5.2.7
- CCS Uniflash, version 3.4.0.00003

When the installer executes with the default settings, the software is installed under *C:\Program Files (x86)\Texas Instruments\TM4C\TM4C129POEAUDIO-1.0*. There are three directories and one executable that the installer will create in the path.

6.1 Common Drivers for the Embedded Application

The folder *appdrivers* contains the driver files for the main application.

6.2 Voice Call Demonstration Project

The folder *poe_voice_call_demo* contains the Code Composer Studio™ (CCS) project for the main application. The folder also has the binary file that can execute on the hardware if no changes are required in the application code.

6.3 Opus Library Project

The folder *opuslib* contains the CCS project for the Opus Voice Codec. When the user compiles the project, it generates the library file *poe_voice_call_demo* for opus encoder and decoder functions.

6.4 NodeAdmin Visual Studio Source Code

The folder *NodeAdmin* contains the visual studio code for the PC application that configures the nodes.

6.5 NodeAdmin Executable

The NodeAdmin executable can be used if no changes required to the source code. There is a shortcut to this application on the start menu for quick launch.

7 Test Setup

This section elaborates on how to import, build, and run the application firmware for voice call.

7.1 Create a Clean Workspace in CCS

Before building the examples, the user must create a new workspace to ensure a clean build of the project.

1. Start CCS.
2. Click on *File*. In the dropdown menu, navigate to *Switch Workplace* then to *Others*.
3. In the popup box, provide the name of the new workspace. Figure 19 shows an example workspace named *workspace_TIDMTM4C129POEAUDIO*. Click *OK*.

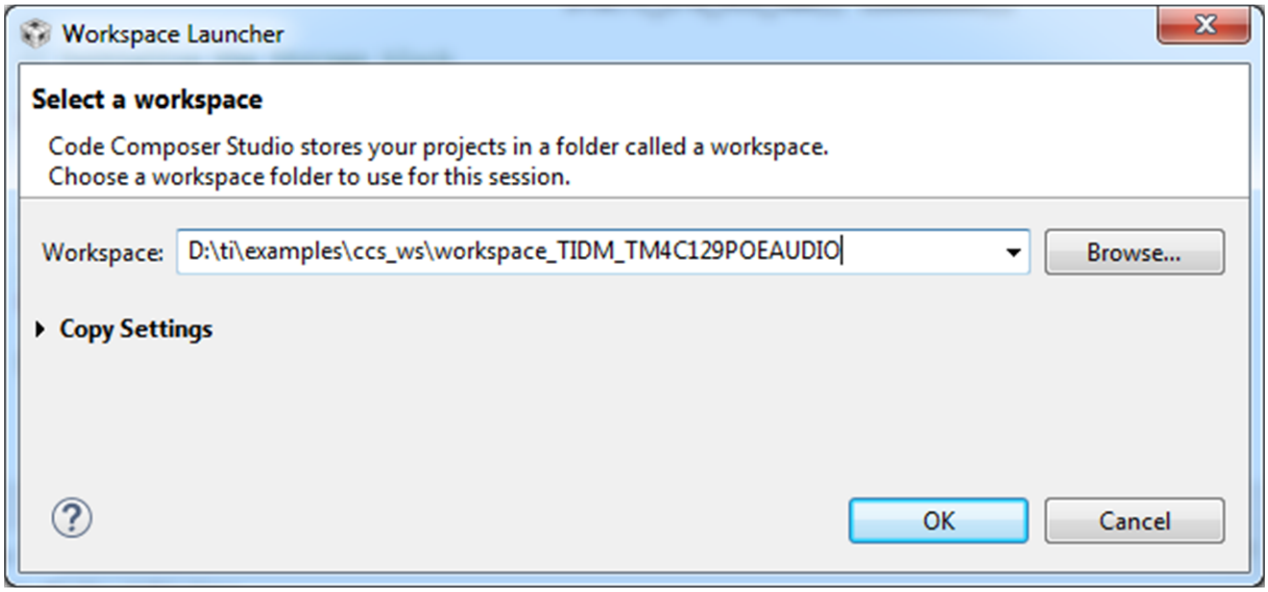


Figure 19. Create a New Workspace

7.2 Import and Build the Projects in CCS

To compile the example code, first build the opus audio codec library. The example code uses the pre-compiled output of the opus audio codec library during the linker phase.

1. In the workspace created (see Section 7.1), select *File* then select *Import*. The *Import* window will be displayed.
2. Expand *Code Composer Studio* for more options, then click *CCS Projects*. Click on *Next* (see Figure 20).

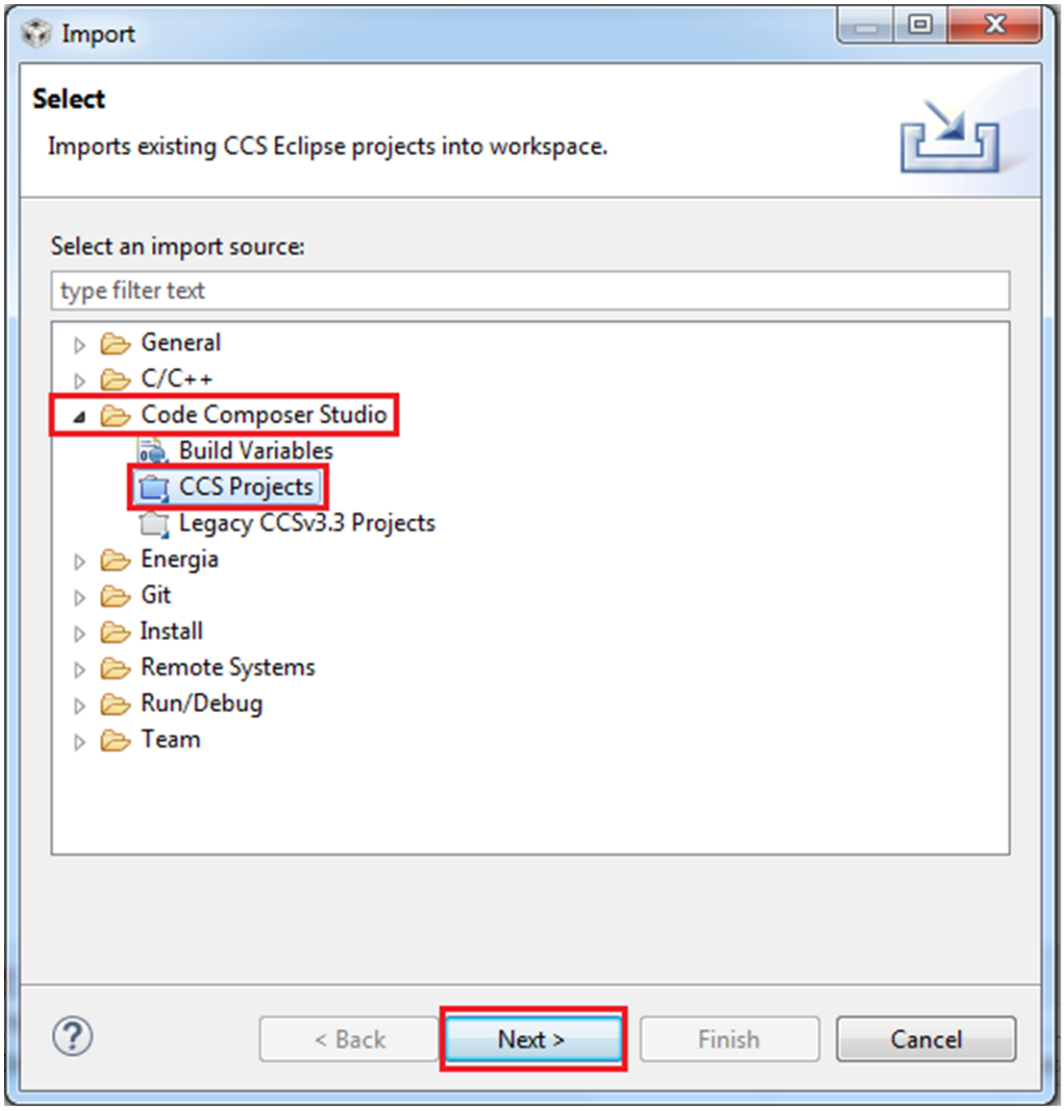


Figure 20. Import the Projects

3. Click the *Browse* button to the right of *Select search-directory:* field and navigate to the directory where the project collateral installed (see Section 6). Select all of the projects listed in the *Discovered projects:* pane. Make sure to check the boxes to the left of *Automatically import referenced projects found in the same search-directory* and *Copy projects into workspace* (see Figure 21). Click the *Finish* button. This step imports the examples into CCS.

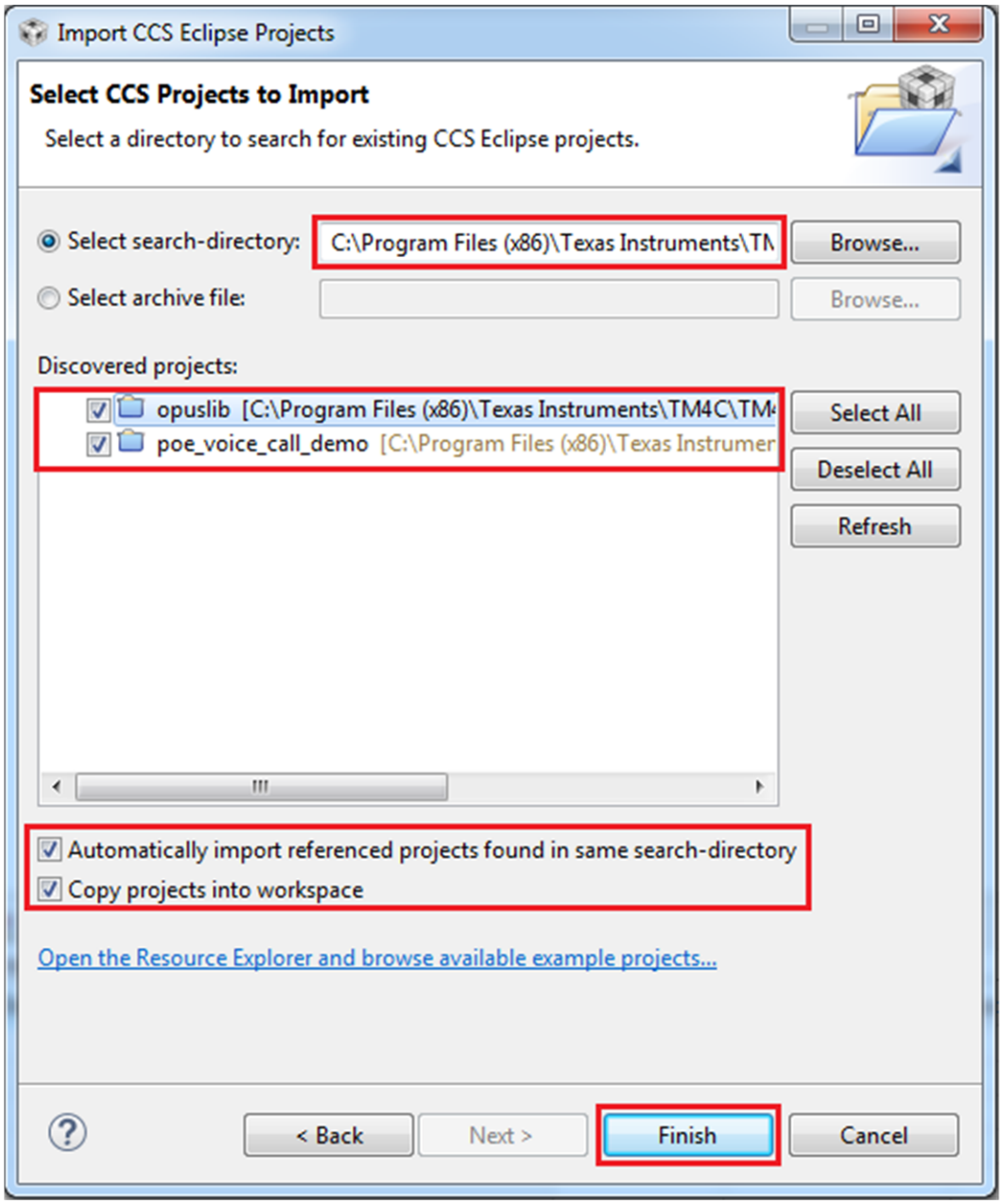


Figure 21. Import the Projects

4. Before building the project *opuslib*, make sure that the variable OPUS_ROOT corresponds with the correct directory with the extracted opus source code. To view or change the variable, right click on *opuslib* and click on *Show Build Settings*. In the pop-up window, click on *Build*. Click on the tab *Variables* to view the value of OPUS_ROOT. The path must match the extracted opus source code path on the machine. If different, select the variable and click on *Edit…* to change the path (see Figure 22).

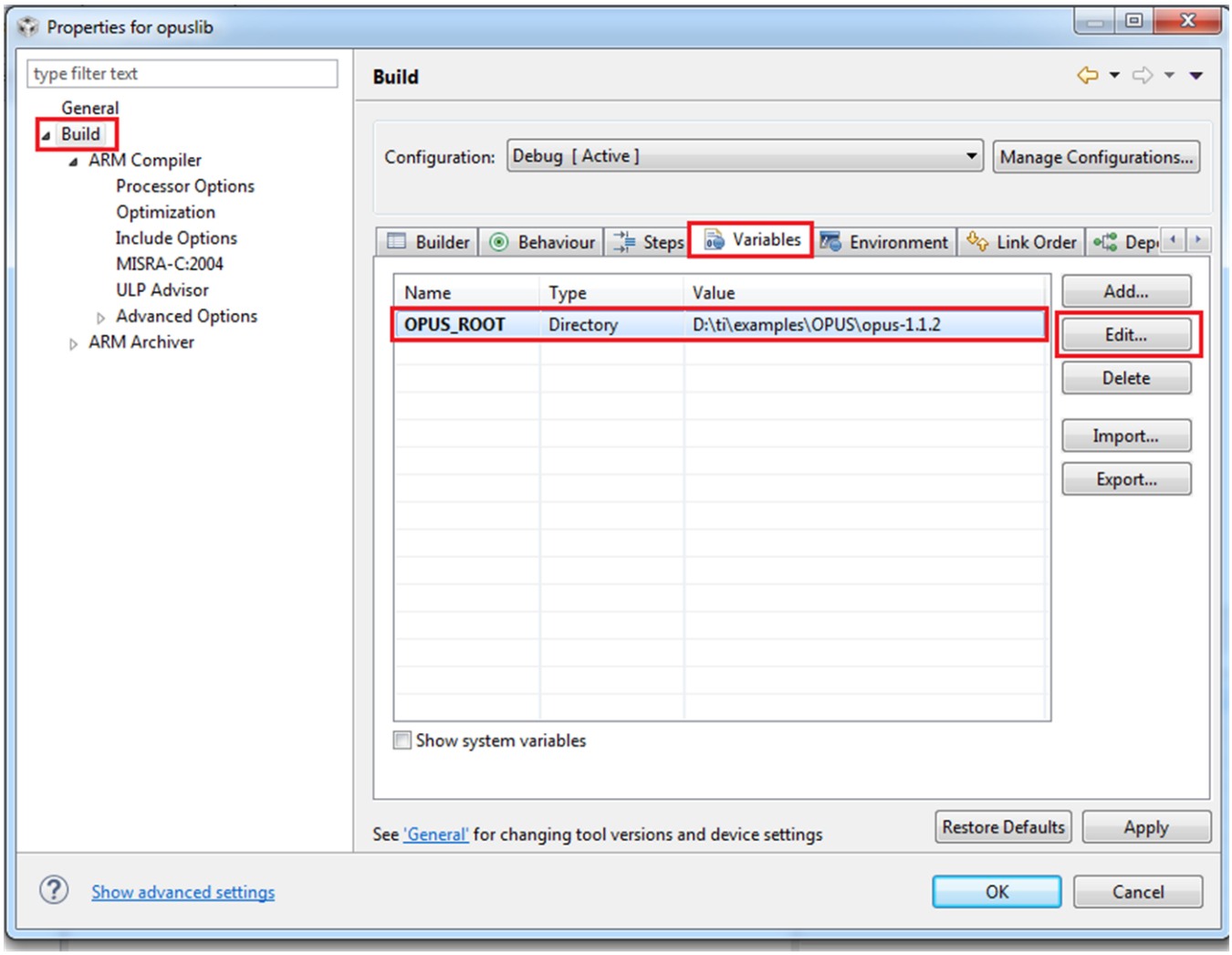


Figure 22. Update the OPUS_ROOT Variable

5. Right-click the opuslib project in Project Explorer and click *Build Project*. If building the library for the first time, it will take a few minutes. After the compilation is successful, the CCS console must display the message in Figure 23.

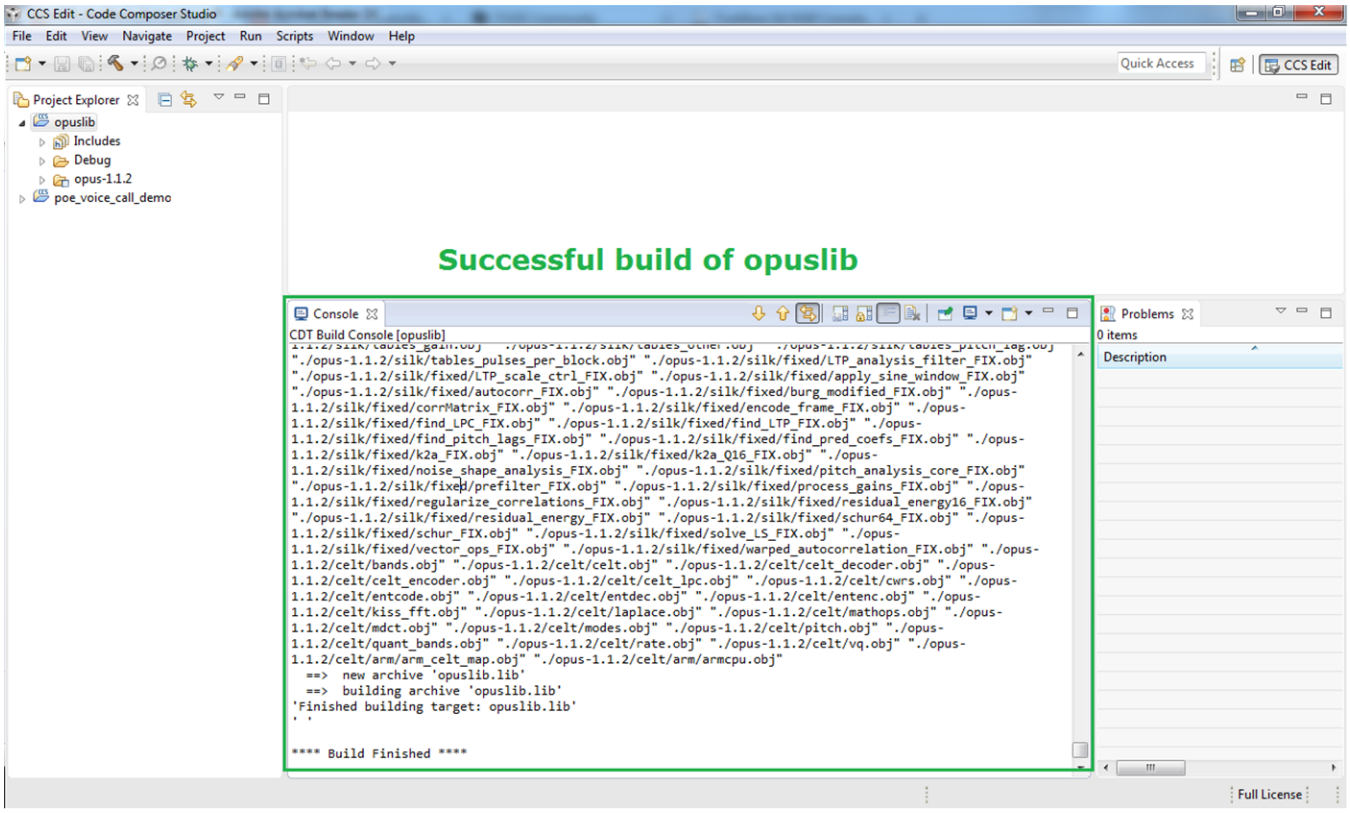


Figure 23. Compiling Opuslib Project

6. Before building the poe_voice_call_demo project, ensure that the variables OPUS_ROOT, SW_ROOT, and TID_ROOT point to the correct directories. The variable OPUS_ROOT must point to the path with the extracted opus source code. The variable SW_ROOT must point to the path with the installed TivaWare 2.1.3. The variable TID_ROOT must point to the path with the installed project collateral. Right-click the poe_voice_call_demo project in Project Explorer and click *Build Project*. After the compilation is successful, the CCS console must display the message in Figure 24.

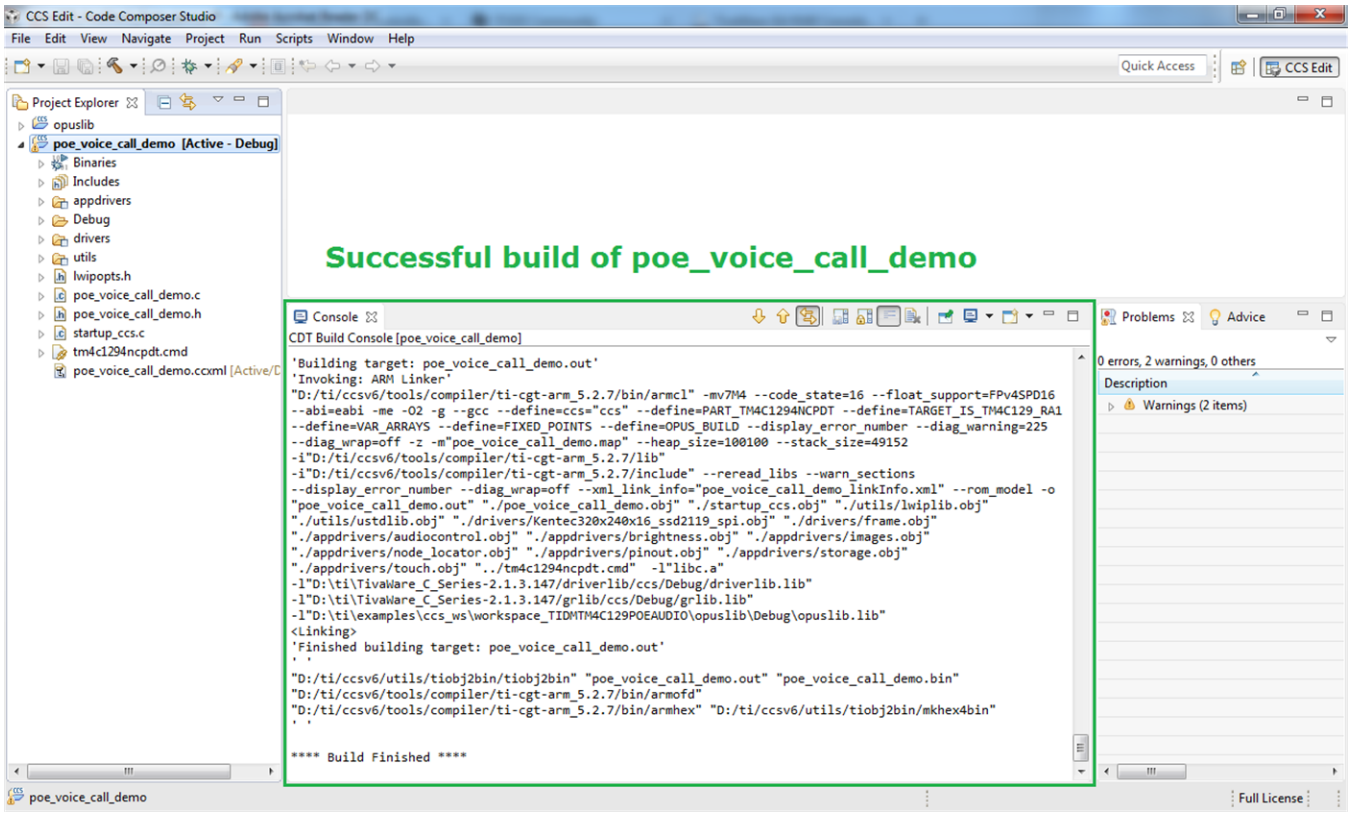


Figure 24. Compiling Poe_voice_call_demo Project

7.3 Programming the MAC Address and Application Binary

The user must start the CCS Uniflash application as the MAC address and the application must program to the MCU.

1. Click on *File* then click on *Open Configuration*.
2. Use the *Browse* button to navigate to the poe_voice_call_demo location and select the file poe_voice_call_demo.ccxml (see Figure 25). Click *OK*.

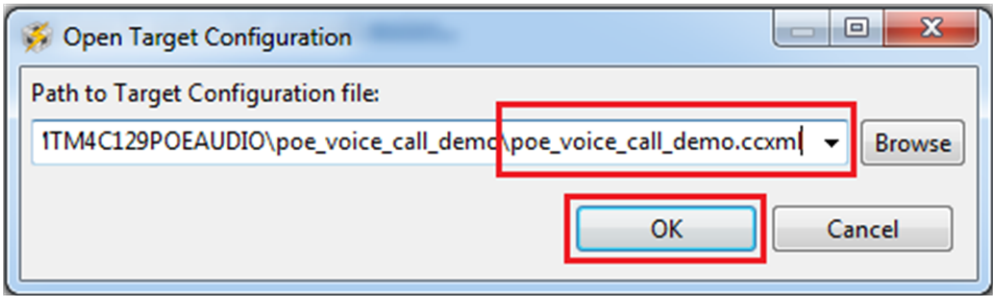


Figure 25. Open-Target Configuration

3. Power up the board to prepare the programming of the MAC address (see Section 5.5). Expand *Flash Settings*, and click on *User Register programming*. To program the MAC address, enter the MAC address per user allocation, select the check box to the left of *Commit MAC Address*, and then click *Program MAC Address* (see Figure 26).

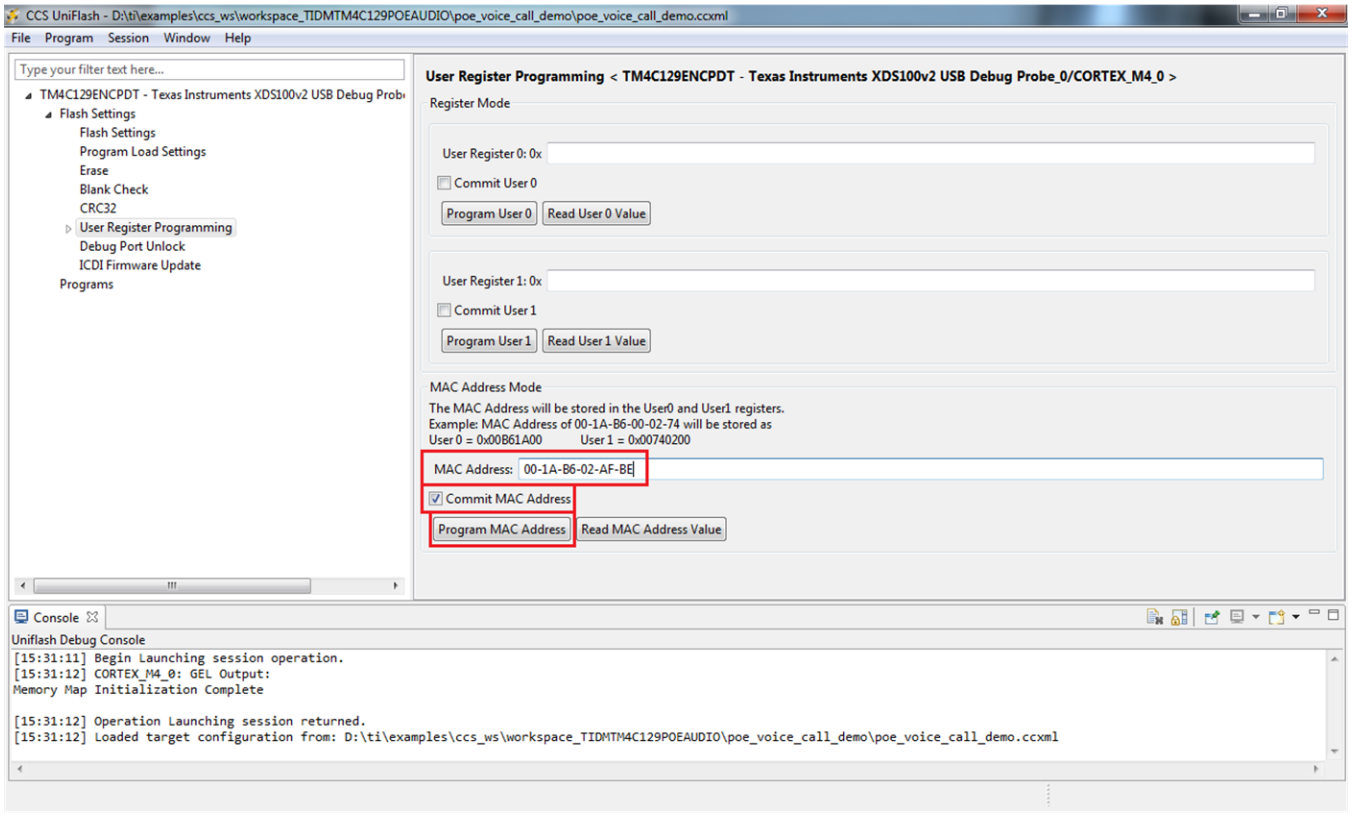


Figure 26. Program the MAC Address

4. Program the application binary to the MCU. In the main menu click on *Program* then click *Load Binary*. In the pop-up box, browse to the path of the generated bin file from Section 7.2. Select the *Start Address* as 0x0. Click *OK*. See Figure 27 for more details.

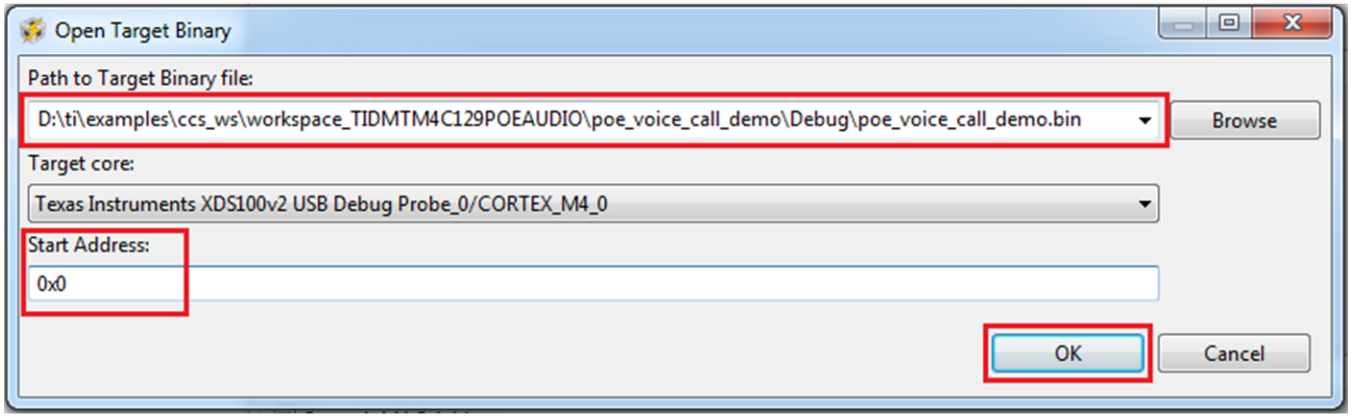


Figure 27. Program the Application Binary

5. Repeat Steps 3 and 4 to program the remaining boards. Ensure the MAC address is unique to each device.

7.4 Configuring the Node Using the NodeAdmin Utility

Now that the MCUs programmed correctly, remote the debugger and the Ethernet cable to power down the board. Place the audio board and LCD BoosterPack as shown in Section 5.2 and Section 5.3. Connect the Ethernet cable to power up the board. Once the boards are powered up, the application binary gets the IP address from the DHCP server on the network. The application powers to the default state requesting for a configuration of the node number as shown in Figure 28.

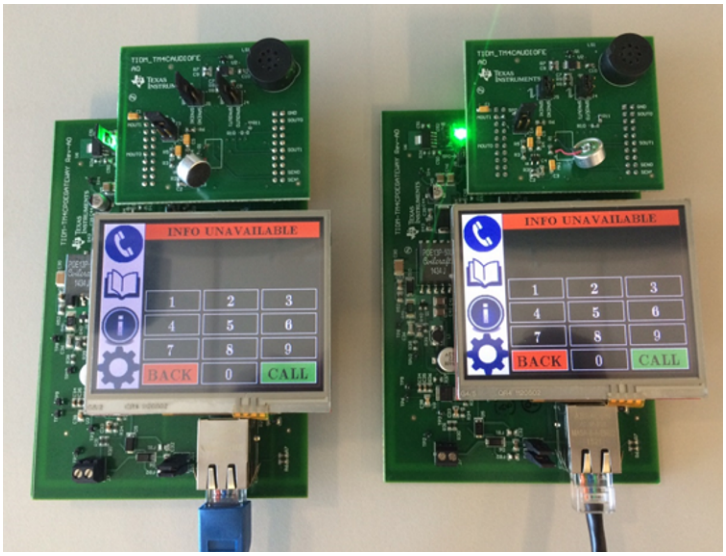


Figure 28. First Power Up of the Design

The user must configure the node number using the NodeAdmin.exe application provided in the software. Touch *INFO UNAVAILABLE* on the LCD panels to navigate to the information window as shown in Figure 29.

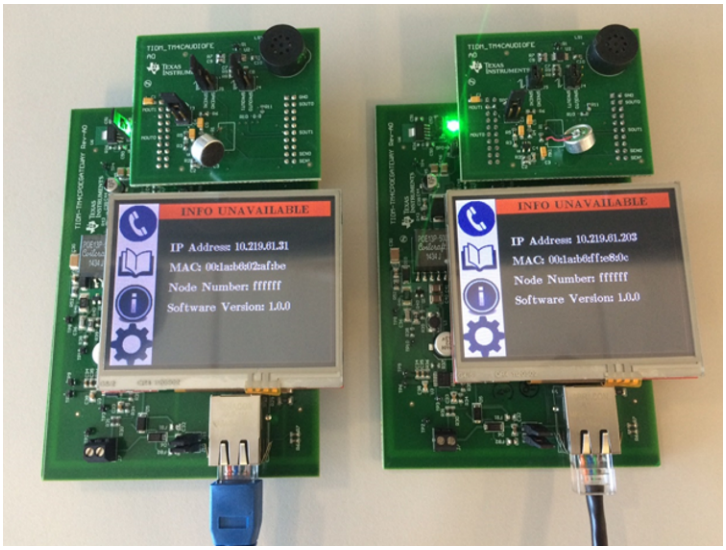


Figure 29. Navigate to the Information Window

To update the node number for each of the boards, follow these instructions:

1. Launch NodeAdmin.exe. The application will send discover packets over the same subnet network to get the MAC and IP address for each node. This information will be displayed in the NodeAdmin application in Figure 30. Since the node numbers are not configured, the Node Number column shows FFFFFF.

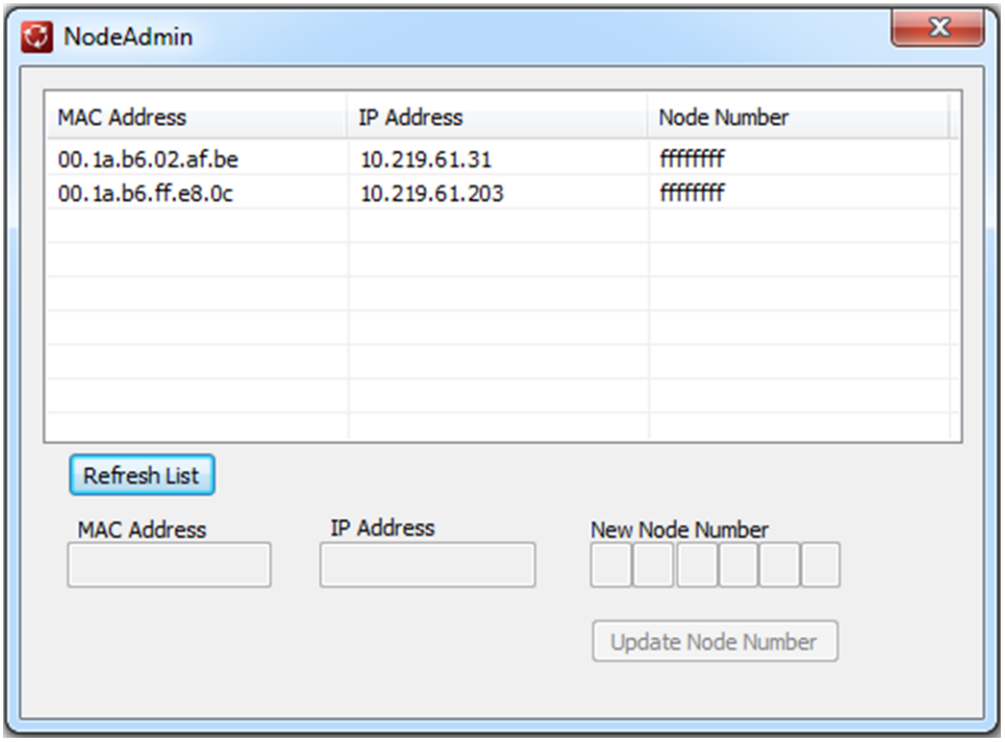


Figure 30. NodeAdmin Launch View

2. Double click on the row corresponding to the board desired. This action will automatically populate the MAC Address and IP Address field in the GUI. The field *New Node Number* will be available for edit as shown in Figure 31. Give the new node number and click on *Update Node Number*.

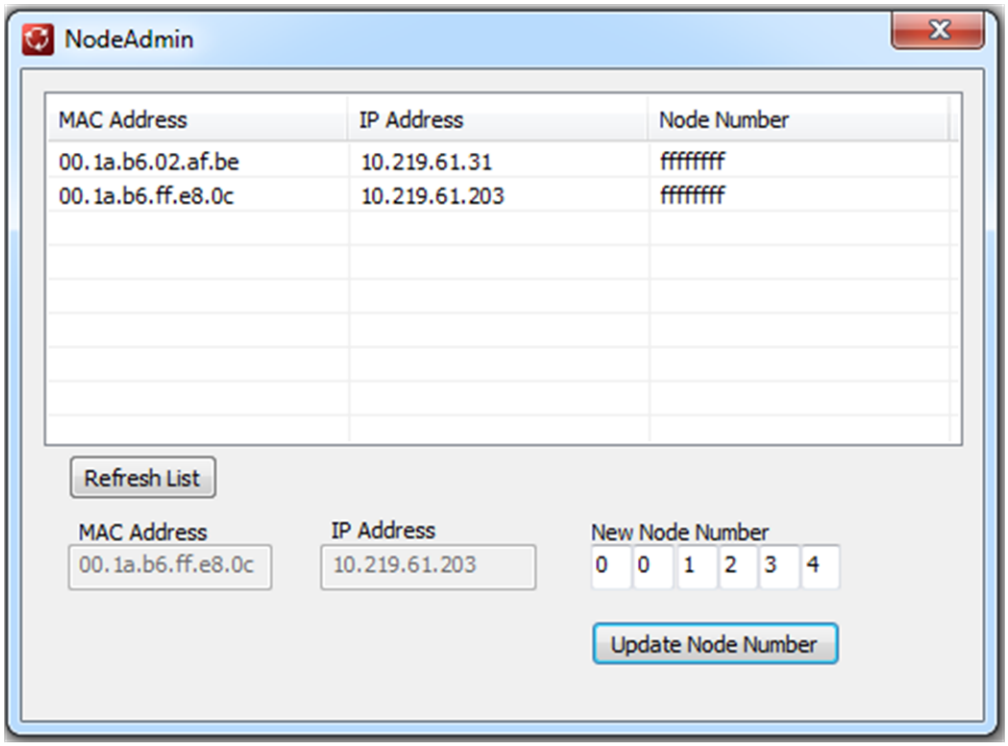


Figure 31. NodeAdmin Update View

3. The device corresponding to the MAC address and IP address will program with the new node number and reboot. After reboot, the device will use the new assigned node number, see Figure 32, for placing and accepting calls. The status of the updated board will now show the assigned node number.

Figure 32. Board Status on Node Number Update

4. Click *Refresh List* in the NodeAdmin application (see Figure 33). This updates the status of the boards. Repeat Steps 2 and 3 to program the remaining boards.

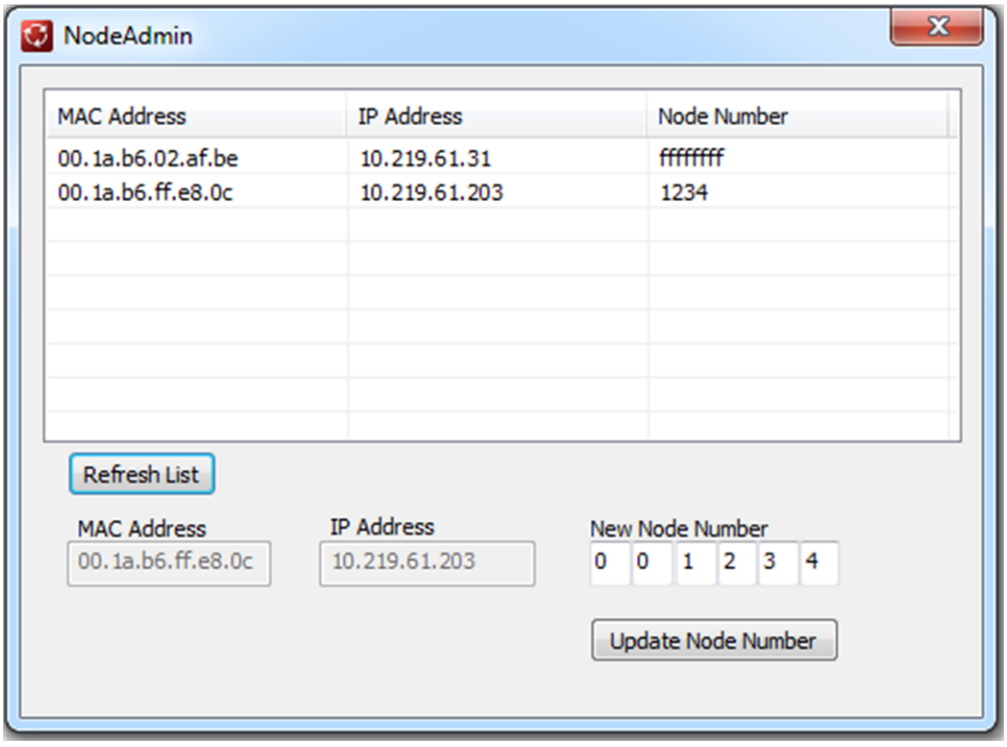


Figure 33. NodeAdmin Refresh View

7.5 Placing and Accepting a Call

Once all boards have a node number, a call can occur (see Figure 34).

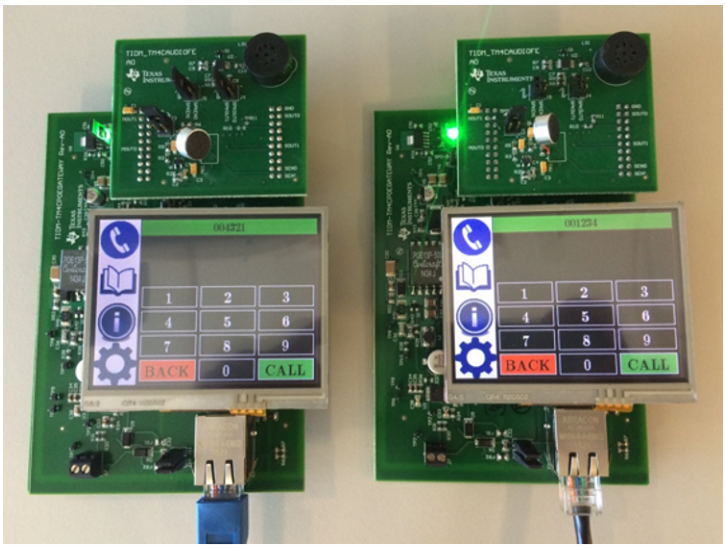


Figure 34. Board Status After Node Number Configuration

The left board's node number is 004321, and the right board's node number is 001234. Dial "004321" using the number pad on the right board then press the *CALL* button (see Figure 35).

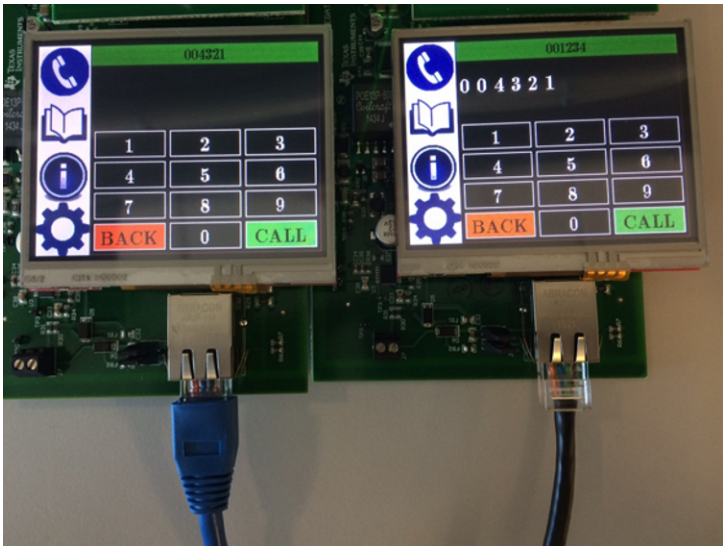


Figure 35. Dialing a Node

Figure 36 shows a call between the two boards. The left board displays the node number that is calling. This board provides the user with two buttons. If the user touches the green button, the call connects, and the two boards can exchange audio data. If the user touches the red button, the call disconnects.

The right board displays the node number that it is calling and provides the user with a single red button. If the user touches the red button, the call disconnects from the calling node side.

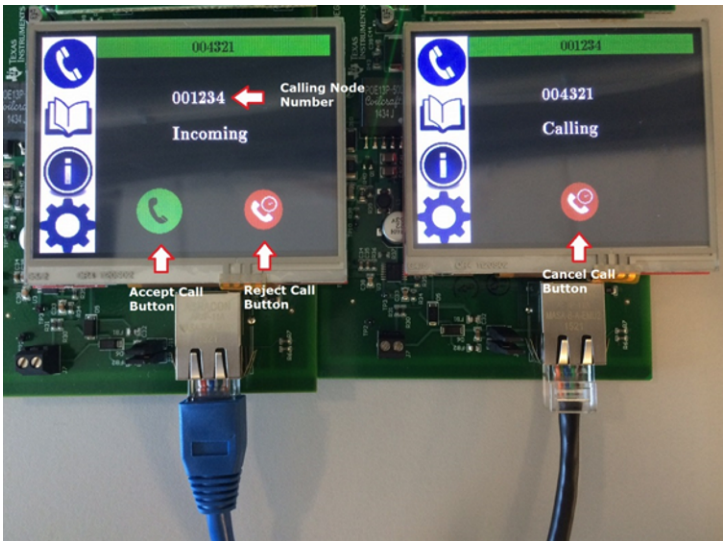


Figure 36. Display Status When Placing a Call

If the call connects, then the two boards display the current call duration and provide the user with a button to disconnect the call. The application also provides a slider to change the in-call volume. Figure 37 shows a connected call.

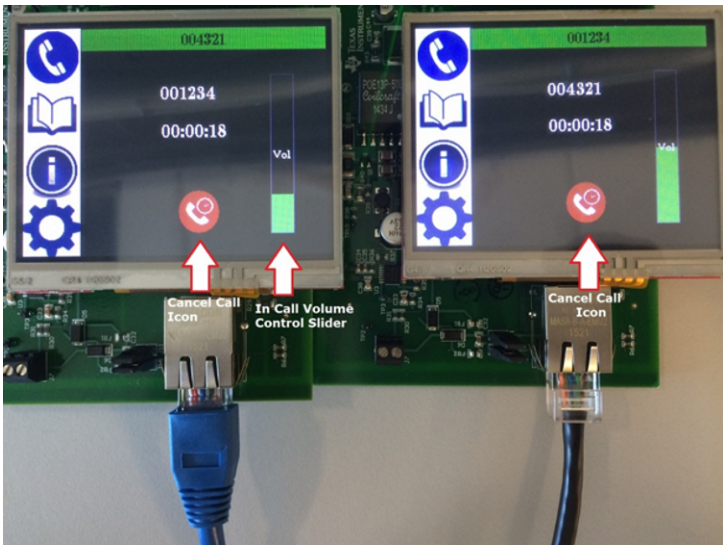


Figure 37. Display Status When Accepting a Call

8 Test Data

This section highlights the collected performance data from a call.

8.1 Network Traffic

Figure 38 shows a captured Wireshark log used to analyze the resulting network traffic from audio-data exchange.

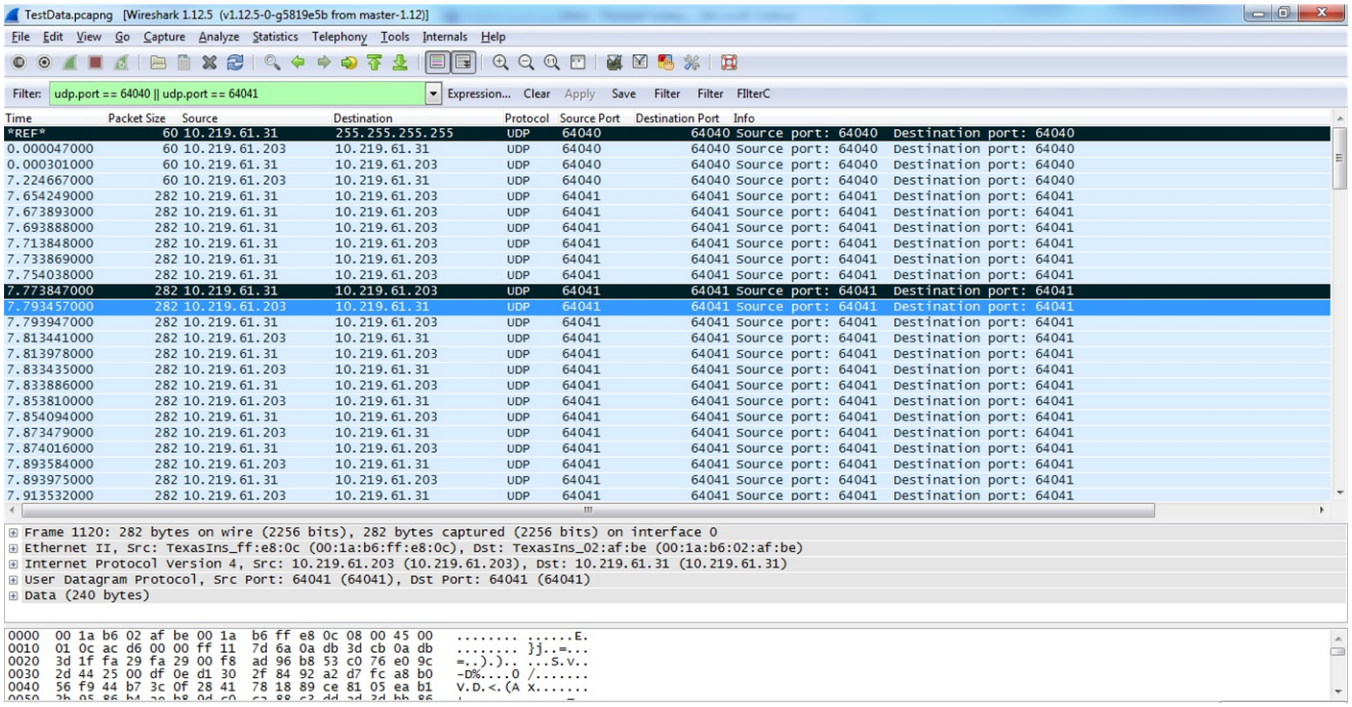


Figure 38. Wireshark Network Capture

The application has been setup with the following parameters:

- Audio sampling rate: 48 kHz
- Bits per sample: 12 bits (ADC resolution)
- Bytes per sample: 2 (as 12 bits are now placed in 16 bits)
- Number of channels: 1 (mono)
- Opus encoder bit rate: 96-KHz constant bit rate
- Opus encoder frame size: 20 ms
- Opus encoder complexity: 0 (lowest) for minimum encoder effort

Based on the setup parameters, a raw PCM bit stream bandwidth requirement for single-node transmission of audio can be calculated as:

- Number of bytes per second = Audio sampling rate × Bytes per sample
- RAW bytes per second = 48e3 × 2 = 96 KB

Once the audio runs through the opus encoder, the Wireshark log shows that 240 bytes of data sends every 20 ms.

- Number of bytes per second = Number of bytes on wire / Opus encoder frame size
- Encoded bytes per second = 240 / 20e-3 = 12 KB

- Network utilization factor improvement = RAW bytes per second / Encoded bytes per second
  = 96 KB / 12 KB
  = 8

The overall network load reduces by a factor of eight for a single call using the opus audio codec versus the RAW PCM bytes being sent.

8.2 Active CPU Usage

Because the opus encoder and decoder require additional CPU to run the required algorithms, CPU usage increases. Figure 39 shows the increase in CPU usage which a GPIO is toggled for the opus encoder and another GPIO toggled for the opus decoder. The encoder and decoder call is performed every 20 ms.

Figure 39 shows the CPU time per audio frame for the encoder is 8.636 ms and for the decoder is 5.395 ms. The total CPU time is 14.031 ms, which is a sum of the encoder and decoder per audio frame CPU time. Thus, total CPU usage is 70.155% (14.031 ms / 20 ms).

This leaves about 30% for the CPU to perform other housekeeping tasks, like display updates, reinitializing the DMA channels, and running the lwIP stack.

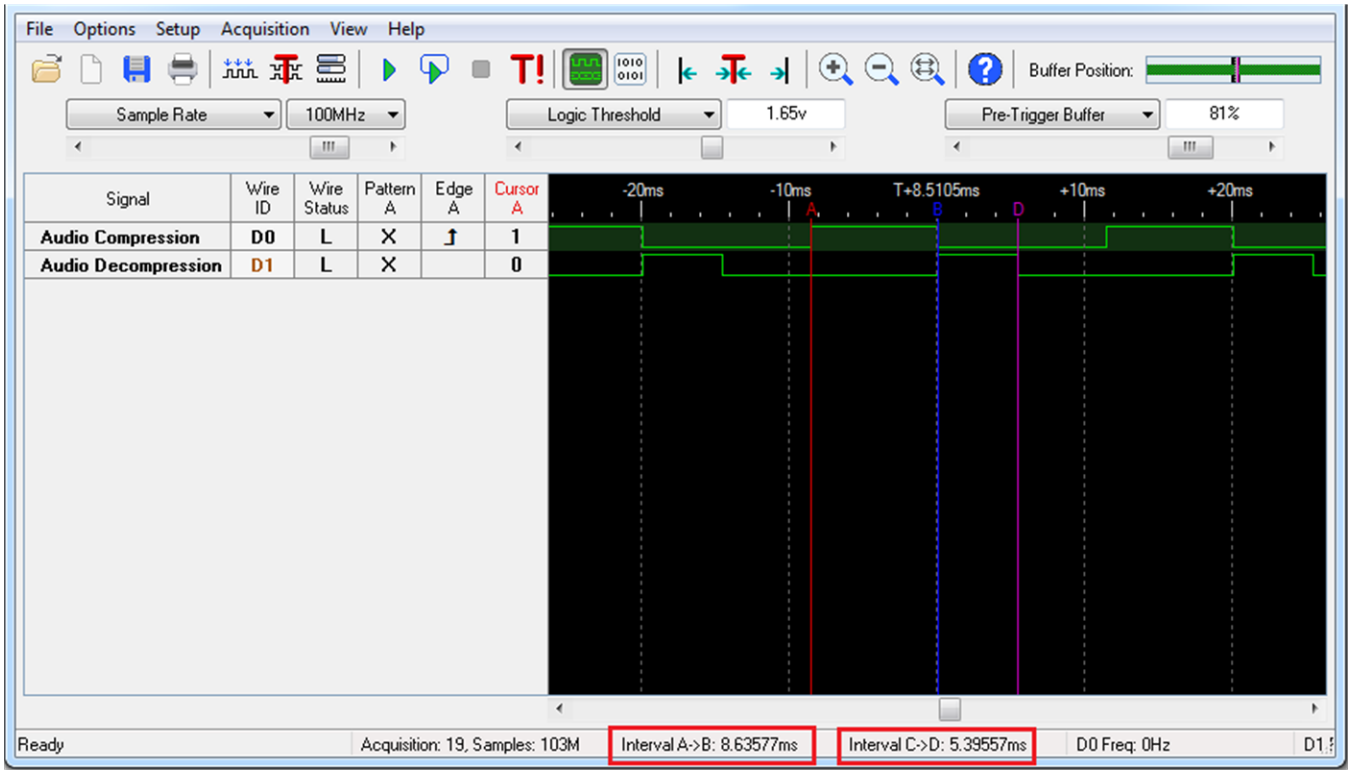


Figure 39. CPU Usage for Audio Codec

9 Design Files

9.1 Schematics

To download the schematics, see the design files at TIDM-TM4C129POEAUDIO.

9.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDM-TM4C129POEAUDIO.

9.3 Layout Prints

To download the layer plots, see the design files at TIDM-TM4C129POEAUDIO.

9.4 Altium Project

To download the Altium project files, see the design files at TIDM-TM4C129POEAUDIO.

9.5 Gerber Files

To download the Gerber files, see the design files at TIDM-TM4C129POEAUDIO.

9.6 Assembly Drawings

To download the assembly drawings, see the design files at TIDM-TM4C129POEAUDIO.

10 Software Files

To download the software files, see the design files at TIDM-TM4C129POEAUDIO.

11 References

1. Texas Instruments, *Implementing OPUS Voice Codec for TM4C129x Device*, Application Report (SPMA076)
2. Texas Instruments, *Single-Supply Electret Microphone Pre-Amplifier Reference Design*, TI Design, (TIDU765)
3. Opus Interactive Audio Codec
4. Netgear, ProSAFE 8-Port Gigabit Smart Switch with PoE and 2 fiber SFP ports (GS110TP)

12 About the Author

AMIT ASHARA is an application engineer and Member Group technical staff member at Texas Instruments. He works on developing applications for the TM4C12x family of high performance MCUs. Amit brings his extensive experience in high-speed digital and MCU system-level design to this role. Amit earned his bachelor of engineering (BE) from the University of Pune, India.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ('TI") reference designs are solely intended to assist designers ("Designer(s)") who are developing systems that incorporate TI products. TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.

TI's provision of reference designs and any other technical, applications or design advice, quality characterization, reliability data or other information or services 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 reference designs or other items.

TI reserves the right to make corrections, enhancements, improvements and other changes to its reference designs and other items.

Designer understands and agrees that Designer remains responsible for using its independent analysis, evaluation and judgment in designing Designer's systems and products, and has full and exclusive responsibility to assure the safety of its products and compliance of its products (and of all TI products used in or for such Designer's products) with all applicable regulations, laws and other applicable requirements. Designer represents that, with respect to its applications, it has 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. Designer agrees that prior to using or distributing any systems that include TI products, Designer will thoroughly test such systems and the functionality of such TI products as used in such systems. Designer may not use any TI products in life-critical medical equipment unless authorized officers of the parties have executed a special contract specifically governing such use. Life-critical medical equipment is medical equipment where failure of such equipment would cause serious bodily injury or death (e.g., life support, pacemakers, defibrillators, heart pumps, neurostimulators, and implantables). Such equipment includes, without limitation, all medical devices identified by the U.S. Food and Drug Administration as Class III devices and equivalent classifications outside the U.S.

Designers are authorized to use, copy and modify any individual TI reference design only in connection with the development of end products that include the TI product(s) identified in that reference design. HOWEVER, 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 published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of the reference design or other items described above 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 REFERENCE DESIGNS AND OTHER ITEMS DESCRIBED ABOVE ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, 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 DESIGNERS AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS AS DESCRIBED IN A TI REFERENCE DESIGN 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 THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TI's standard terms of sale for semiconductor products (http://www.ti.com/sc/docs/stdterms.htm) apply to the sale of packaged integrated circuit products. Additional terms may apply to the use or sale of other types of TI products and services.

Designer will fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of Designer's non-compliance with the terms and provisions of this Notice.

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