Wireless Microphone with USB Interface

By Michael Burns

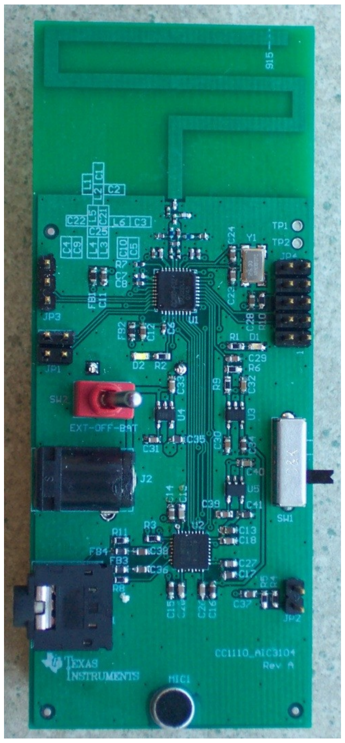
Keywords

- *CC1110*
- *CC1111*
- *CC2510*
- *CC2511*
- *USB Audio*
- *Wireless Microphone*

Introduction

This Application Note is intended to show how the CC111x and CC251x series of Low-Power SoC (System-on-Chip) RF Transceivers can be used to provide a low cost audio to USB interface. Two printed circuit board assemblies (PCBAs) are used for this purpose; a microphone PCBA based on the CC1110 and TLV320AIC3104 Low Power Stereo Audio Codec, and a standard CC1111 'USB dongle'. The CC1111 is similar to the CC1110, with the addition of a full-speed USB 2.0 compatible interface. Audio is sampled at a 16 kHz sample rate, and 16 bit resolution is maintained.

While the design described is based on the C1110 and CC1111 transceivers and operates in the 900 to 928 MHz ISM bands, it is easily modified to work with the CC2510 and CC2511 transceivers, which operate in the 2.4 GHz ISM band. Project collateral discussed in this application note can be downloaded from http://www.ti.com/lit/zip/SWRA368.



CC1110_AIC3104 Wireless Microphone

CC111 USB Dongle

Table of Contents

1 Abbreviations

ADC	Analog to Digital Converter
AGC	Automatic Gain Control
AUX	Auxiliary
Codec	Coder-Decoder
CRC	Cyclic Redundancy Check
DAC	Digital to Analog Converter
dB	Decibel
dBm	Decibel (referenced to one milliwatt)
DMA	Direct Memory Access
LDO	Low Drop Out
EVM	Evaluation and Verification Module
FCC	Federal Communications Commission
FSK	Frequency Shift Keying
GFSK	Gaussian Frequency Shift Keying
GHz	Gigahertz
GPIO	General Purpose Input/Output
I^2C	Inter-Integrated Circuit
I^2S	Integrated Interchip Sound
ISM	Industrial, Scientific, Medical
ISR	Interrupt Service Routine
kbps	kilobit per second
kHz	kilohertz
LED	Light Emitting Diode
MCLK	Master Clock
MHz	Megahertz
ms	Millisecond
MSK	Minimum Shift Keying
mVpp	millivolts peak-to-peak
mW	milliwatt
PCB	Printed Circuit Board
PCBA	Printed Circuit Board Assembly
PLL	Phase Locked Loop
RF	Radio Frequency
SOC	System-On-Chip
TX	Transmit
µs	microsecond
USB	Universal Serial Bus
VRMS	Volts RMS (Root Mean Square)

2 Microphone PCBA - Overview

The microphone PCBA is based on the TLV320AIC3104 Codec [5] and the CC1110 SoC RF transceiver [1]. A block diagram is shown in Figure 5 and Figure 6. The ADC sample rate is 16 kHz with 16 bit ADC resolution. This results in an audio bit rate of 256 kbps. The CC1110 is used in normal mode, so that the built in packet handling hardware features of the CC1110 can be used.

The CC1110's packet format is shown in Figure 1. It includes a four byte preamble, four SYNC bytes, a one byte address field (used to identify the transmitting device), the ADC samples, and two CRC bytes. For the sake of compatibility with the SmartRF™ Studio software [6], a one byte length field is also included. Packet length is limited to 255 bytes, not including the preamble, SYNC, and CRC bytes.

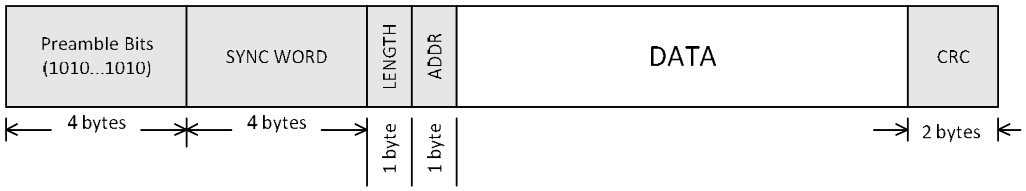


Figure 1. CC11xx Packet Format

Note that the additional 12 bytes (4 preamble bytes + 4 sync bytes + length byte + address byte + 2 CRC bytes) required by the packet handler results in the RF data rate (rate at which data is transmitted 'over the air') being greater than the audio data rate (rate at which the ADC is sampled).

The USB audio protocol requires a 48 kHz sample rate, requesting 48 16-bits samples every millisecond. Since the microphone PCBA's sample rate is 16 kHz (16 samples every millisecond), each sample is repeated three times and transferred to the host controller. Since packet overhead (12 bytes) is fixed, it is more efficient to send as much data as possible per packet (i.e., maximize packet length). However, the longer the packet, the more likely that interference can cause packet errors. As a compromise, 96 ADC samples (192 bytes) are sent in each data packet. A RF data rate of 300 kbps is used. GFSK modulation was selected, based on the recommendations of SmartRF Studio [6]. Given the sample rate of 16 kHz, a packet is sent every 6 milliseconds. At a RF data rate of 300 kbps, each packet requires 5.44 ms to send (((192 + 12)·8)/300000). The time required for the transmitter to transition from the Idle state to the TX state (75.2 µs, see the "State Transition Table" in [1]) must be added to this. This leaves approximately 6000 – (5440 + 75) or 485 µs of idle time between packets.

A four channel frequency hopping algorithm is employed. The channel (and therefore the transmit carrier frequency) is changed (incremented modulo 4) for every packet in a repeating pattern of four frequencies. In addition, the PCBA includes a three position "band" switch. The four channels in each of the three bands are unique. Frequency hopping is used to decrease the probability of packet loss due to multipath and/or intentional radiator interference.

Band	Active_Channel_Index			
	0 (time = n)	1 (time = n + 6 ms)	2 (time = n + 12 ms)	3 (time = n + 18 ms)
1	0 (902.5 MHz)	20 (907.5 MHz)	4 (903.5 MHz)	16 (906.5 MHz)
2	74 (921.0 MHz)	98 (927.0 MHz)	78 (922.0 MHz)	94 (926.0 MHz)
3	8 (904.5 MHz)	82 (923.0 MHz)	12 (905.5 MHz)	90 (925.0 MHz)

Table 1. Frequency Hopping Channel Allocation

In Table 1, the transmitter carrier frequency is given as 902.5 MHz + (channel number·250 kHz), where 902.5 MHz is the base frequency and 250 kHz is the channel spacing.

The TLV3203104 Codec is a very versatile part, containing two ADCs and two DACs. It supports sample rates from 8 to 96 kHz. Included is an optional AGC, with programmable parameters (maximum gain, target output). For this design, only the left ADC is enabled. The digital control registers are interfaced via an I^{2}C bus, created from two GPIO pins of the CC1110. Digital audio is interfaced to the CC1110 via an I^{2}S bus.

3 Microphone PCBA - Detailed Description

Refer to Figure 5 and Figure 6. The Microphone PCBA contains only five active devices; the CC1110 RF transceiver [1], the TLV320AIC3104 Codec [5], two 3.3 V LDO linear regulators (TPS73033) [8], and a 1.8 volt LDO linear regulator (TPS73018) [8]. Power is supplied by either 3 AAA batteries or an external 5 V supply. A small condenser style microphone is installed on the PCBA, along with a jack for an external microphone.

The TLV320AIC3104 requires a Master Clock (MCLK), which is derived from Timer 4 of the CC1110. The frequency of this clock signal (13 MHz) is one half of that of the CC1110's 26.0 MHz crystal. The digital logic of theTLV320AIC3104 requires a 1.8 V supply, which is derived from the TPS73018 LDO regulator. The I/O pins of the TLV320AIC3104 are supplied from the VCCD LDO regulator, along with the CC1110. While not strictly necessary, the analog supply pins of the TLV320AIC3104 are fed from a separate 3.3 V regulator. As the CC1110 transitions from idle to transmit mode and back, the current drawn from the regulator increases (decreases) by about 16 mA. This results in a 'glitch' in the voltage supplied to the CC1110 ("VCCD" in Figure 5) of approximately 23 mVpp, as shown in Figure 2.

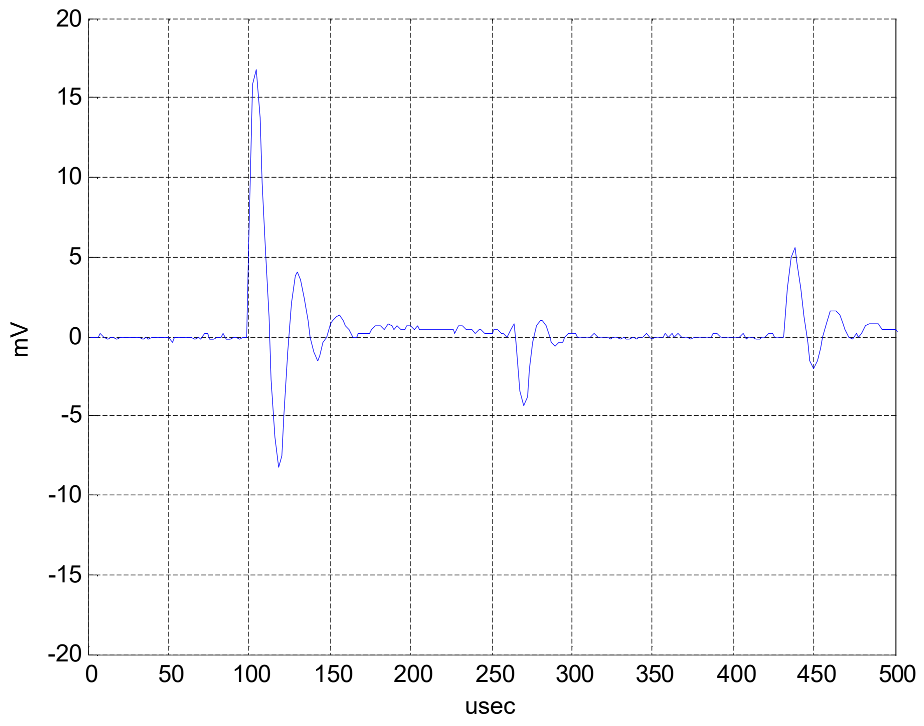


Figure 2. VDD 'Glitch' on Entering TX Mode

By using a separate regulator ("VCCA" in Figure 5), this glitch will not introduce noise into the audio.

To reduce the potential for RF interference, the internal (or external) microphone is connected to the MIC1LP and MIC1LM pins of the Codec using differential input mode. A condenser (electrostatic) microphone requires a bias voltage; this is provided by the Codec at the MICBIAS pin. An external high level (up to 2 VRMS) signal can be used in place of a microphone, via the "LINE2L" input pin. The line input is enabled by installing a jumper on the "Select Aux In" pins of the "Option Jumpers" header.

The TLV320AIC3104 Codec includes an AGC (Automatic Gain Control). This prevents overloading the ADC on loud sounds (and the resulting distortion), while increasing microphone gain on soft sounds so that they may be more readily heard. Many AGC parameters are programmable, such as the target level, maximum gain, and attack and decay times. The AGC feature of the TLV320AIC3104 Codec can be enabled or disabled via the "ENA AGC" jumper pins of the "Option Jumpers" header.

The PCB includes a meandered monopole antenna. Since the CC1110 has differential RF input\output pins, a four element passive network (balun) is used to convert the transceivers differential (balanced) RF input\output signals to a single ended (unbalanced) signal. A low pass filter is required to reduce the harmonic content of the transmitter to levels that comply with FCC regulations. This PCBA uses the 'standard' matching network, as described in the CC1110 Reference Design [3].

The CC1110 contains an internal ADC, which is used to monitor the battery voltage. When the battery voltage drops below approximately 3.6 V, the green LED is flashed at a 250 ms rate. If the battery voltage exceeds 3.6 V, the green LED will remain on.

IMPORTANT: *The radiation pattern of the Wireless Microphone card is such that better range is obtained with the PCB antenna oriented in the vertical plane.*

4 USB Dongle - Overview

This application makes use of a CC1111 USB Evaluation Module Kit [2]. The CC1111 uses the same radio as the CC1110, and is set up to listen for a packet containing 96 16-bits ADC samples every 6 ms. As described in Section 2, the USB audio protocol operates at a 48 kHz sample rate and expects 48 16-bits audio samples every millisecond. Since the microphone's sample rate is 16 kHz (i.e., 16 samples per millisecond), each ADC sample is repeated 3 times in order to obtain the required 48 samples per millisecond.

5 Software Description - Microphone

The software program for the microphone is divided into seven program segments as follows:

1) tw_main.c
    - Main program segment.
2) tw_rf.c
    - Radio setup ("initRF")
    - Configuration ("rfConfigRadio")
    - Packet send ("rfSendPacket") routines.
3) tw_dma.c
    - Contains the DMA configuration routines
        - "dmaToRadio" (from the TX buffer to the radio's transmit data register, RFD, using DMA channel 2),
        - "dmaAudio" (from the Codec to the AudioIn buffers using DMA Channel 4),
        - "dmaMemtoMem" (memory-to-memory using DMA channel 0)
    - DMA interrupt service routine
4) Init_peripherals.c
    - Initializes the CC1110's peripherals
        - P0 through P2
        - The I^{2}S controller
        - Timer 1 (generates an interrupt every 250 ms - used to flash the Heartbeat and Low Battery LEDs)
        - Timer 3 (TX timeout)
        - Timer 4 (used to generate Codec MCLK)
5) init_codec.c
    - Initializes the TLV320AIC3104 Codec control registers
6) i2c.c
    - implements a write only I^{2}C interface using two DI/DO pins (I2C_SDA and I2C_SCL)
    - Subroutine I2Cwrite writes "data" into the specified Codec register (page addr, reg_addr)
7) tw_interrupt.c
    - Handles Radio and Timer 1 interrupts.

In addition, there are two 'include' files. "board.h" contains definition statements, option selections, and subroutine prototype declarations. "TLV320AIC3104.h" contains definitions specific to the TLV320AIC3104 Codec.

A flow chart of the Microphone PCBA's 'Main' program is shown in Figure 7. After initializing the I/O ports, I^{2}S controller, Timers, DMA controller, Codec and Radio registers, the main program loop is entered. The main program loop is repeated every 6 milliseconds.
At the beginning of the loop, the next channel is selected and PLL calibration started. The channel selected is based on the 'band' and 'Active_Channel_Index' values. There are three bands (0 - 2), with four channels (0 - 3) in each band (see Table 1). In order to reduce the PLL calibration time from the "full

calibration" value of 735 µs, "charge pump calibration" is disabled. This reduces PLL calibration time to 168 µs. This technique is described in the section "Frequency Hopping and Multi-Channel Systems" in [1]. While the PLL is calibrating, the status of the AGC and AUX Select jumpers is checked. If either has changed state form the previous loop, adjustments are made to the Codec register settings as appropriate. Similarly, the position of the band switch is checked, and the value in variable *band* set accordingly.

Next, a check is made of the *AudioFrameReady* flag. This flag will be set by the DMA Channel 4 ISR every 96 ADC samples (6 ms). If set, DMA channel 0 is used to transfer data from the Codec, using the "inactive" *audioOut* buffer, to the data field of the TX buffer. Once the DMA transfer is complete and PLL calibration has finished, a packet is sent.

DMA channel 4 is driven by the I^{2}S bus and is used to transfer ADC samples from the Codec to the *audioOut* buffers. There are two *audioOut* buffers, the "active" buffer and the "inactive" buffer. Which buffer is active is identified by the value of the *activeOut* variable (0 or 1). The DMA controller will be set up to transfer ADC data into the "active" *audioOut* buffer. Note that the Radio will take data from the "inactive" buffer, when needed. Every 96 ADC samples, the DMA ISR will switch the DMA destination address to the address of the "inactive" buffer and toggle the *activeOut* variable. At the same time, the *AudioFrameReady* flag will be set.

6 Software Description - USB Dongle

The USB Dongle software makes use of the USB interface software library, available from Texas Instruments [5]. Details of this library will not be included in this document. What has been added for this application includes the main program segment "simple_audio_dongle.c". Several additional segments are nearly identical to the segments described in Section 5, with the exceptions noted below.

- tw_rf.c
    - Since this application only receives packets, the "Listen_for_Master" subroutine replaces the "rfSendPacket" subroutine.
- tw_dma.c
    - Contains the DMA configuration routines
        - "dmaFromRadio" (from the radio's receive data register, RFD, to the RX Buffer using DMA channel 1)
        - "dmaToAudioIn" (transfers ADC samples or 'nulls' to the audioIn buffer using DMA channel 3)
        - "dmaToAudioOut" (transfers ADC samples into the USB_fifo_all buffer using DMA channel 4)
        - "dmaToUSBfifo" (transfers data from a buffer into the radios USBF4 register using DMA channel 0)
- init_peripherals.c
    - Timers 3 and 4 are not used, nor is the I^{2}S controller

In addition, the "include" file "twoway.h" contains definition statements, option selections, and subroutine prototype declarations.

Key to understanding the USB Dongle code is an explanation of the various data buffers and the use of the DMA controller's memory-to-memory transfer capability. The DMA controller is capable of copying data from one block of memory into another, or from one source address (byte or word) into a block of memory. The latter feature is used to copy data from a radio register (e.g. RFD) into a buffer or to fill a buffer with zeros (e.g., to mute the audio when a packet is lost). Note that all data must be contained in XDATA memory space. Copying data using DMA is considerably faster than letting the CPU handle it. Figure 3 shows the various buffers and by what means (which DMA channel) data is copied from one buffer to the other.

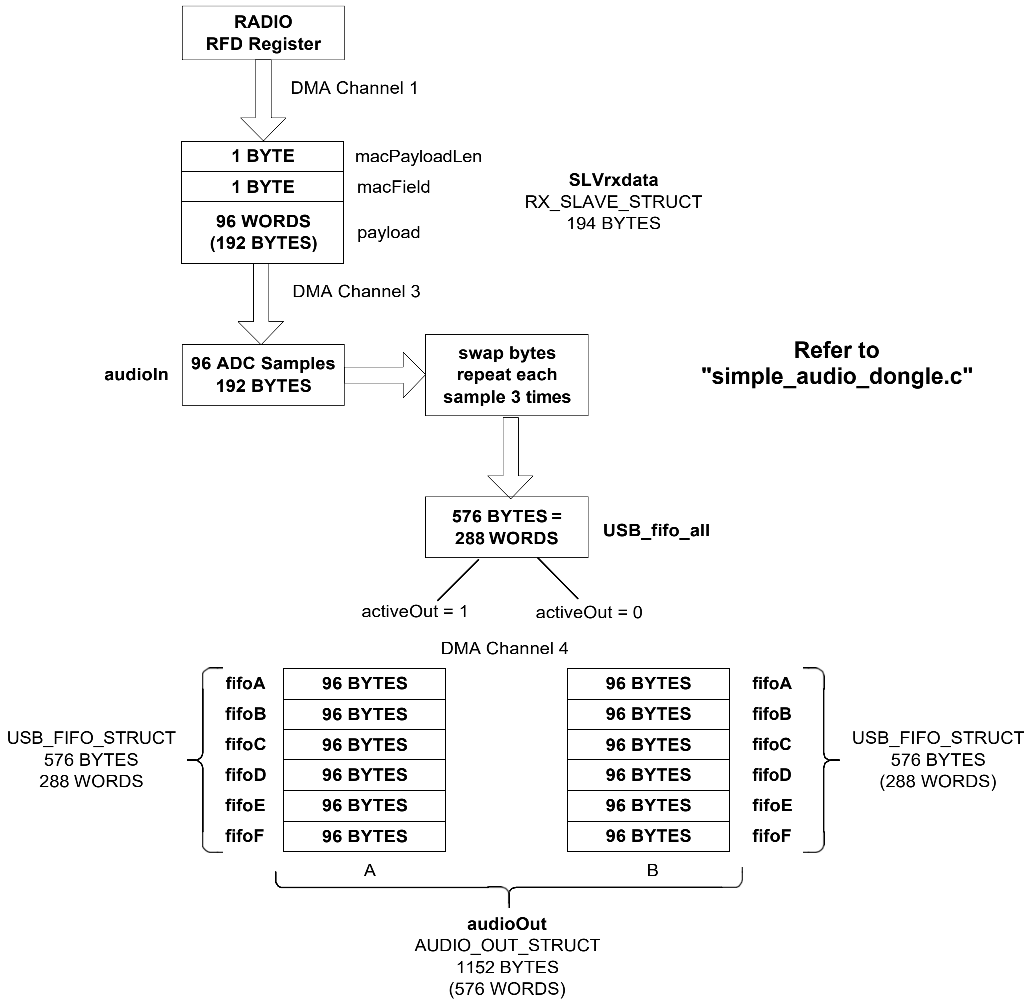
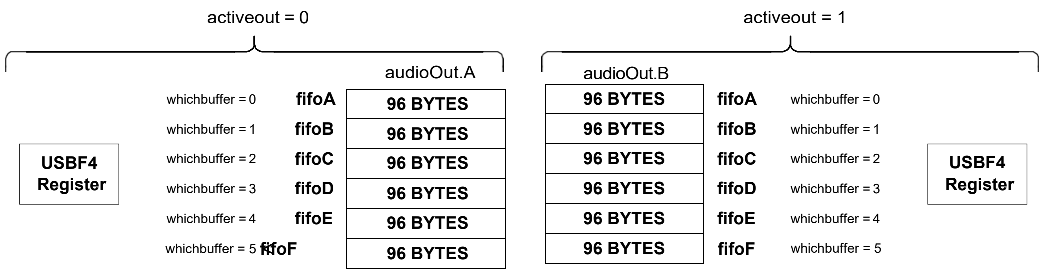


Figure 3. USB Dongle Data Buffers

While the receiver is active, data is copied from the radio's RFD register to the *SLVrxdata* buffer (194 bytes) via DMA Channel 1. The payload field of this buffer will contain 96 16-bits ADC samples. These ADC samples are copied into the *audioIn* buffer (192 bytes) using DMA Channel 3. Note that the two bytes of each ADC sample must be swapped, because of little versus big endianness incompatibility. Since the ADC samples at 16 kHz rate and the host PC expects a 48 kHz sample rate, each ADC sample is repeated three times. The result of this manipulation is placed in the *USB_fifo_all* buffer (576 bytes).

Finally, DMA Channel 0 is used to transfer 48 ADC samples from one of twelve segments of the *audioOut* buffer to the USBF4 register. Every millisecond, the DMA's source address descriptor is updated to point to the next segment (fifoA, fifoB, etc) of the *audioOut* buffer. Every 6 ms, the *activeOut* identifier is toggled. This code is contained in the "usbirqHookProcessingEvents" ISR (see Figure 4).



All transfers use DMA Channel 0. Refer to "usbirqHook ProcessingEvents" in "usb_audio_hooks.c"

Figure 4. DMA Channel 0 to USBF4

A flow chart of the USB Dongle's "Main" program ("simple_audio_dongle.c") is shown in Figure 8. After initializing the I/O ports, Timers 1 and 2, the DMA and USB controllers, and the Radio registers, the main program loop is entered. The main program loop is repeated every 6 ms.

At the beginning of the loop, the "usbAudioProcessEvents" subroutine is invoked. This routine is used to process standard USB requests. If a signal from the Microphone ("Master") has not been successfully received for at least 4 consecutive loop iterations, the "waiting for beacon" code (Figure 9) is executed. This code sets the radio's frequency (channel) to the first entry in the "channel" table of the current band and re-calibrates the PLL. The program will remain in the "waiting for beacon" routine until the microphone's signal is heard. To be heard, a packet must not only be received, but have the proper length and be error free (no CRC errors). Note that for each band, the radio's receiver will remain on for a period of "LISTEN_FOR_BEACON_TIMEOUT" (26 ms). Since this period is greater than 4 times the packet repeat rate (6 ms), the microphone's signal should be heard if it is transmitting in the current "band" setting. Note that the microphone could be using any of three bands, depending on its band switch setting. Once a RF link to the microphone is established (the *waitingforbeacon* flag is false), processing continues.

To avoid timing errors, processing is delayed until Timer 2 (the "Frame Timer") has counted down below the "LISTENFORMASTER" value. The receiver is then turned on and DMA channel 1 enabled. The program will loop until either SYNC is detected or Timer 2 has counted down to below the "SYNCTIMEOUT" value. If a SYNC time out occurs, the radio is idled and DMA channel 1 aborted.

As explained previously, the ADC samples contained in the *SLVrxdata* buffer must be manipulated (bytes swapped and samples repeated) and copied into the *USB_fifo_all* buffer. In addition, the data in the *USB_fifo_all* buffer must be copied into either the *audioOut_A* buffer (if *activeOut* equals zero) or the *audioOut_B* buffer (if *activeOut* equals one), using DMA channel 4. This is done during the time period from when SYNC is detected until the packet is complete.

After the DMA transfer, a loop is entered, waiting for either one of two events to occur;

1) the packet completes (Radio goes to the IDLE state) or
2) the Timer 2 count reaches 0.

In the latter case, the receiver is idled, DMA channel 1 aborted, and *rxStatus* set to PKT_TIMEOUT. In the former case (a packet is received), the loop is exited with *rxStatus* set to SYNC_DETECTED. If *rxStatus* equals either SYNC_TIMEOUT or PKT_TIMEOUT, the packet was not received; *rxPacketStatus* is set to TIME_OUT_ERROR and T2 is allowed to reach a value of END_OF_FRAME (6 ms after Timer 2 was started). If the packet was received, a check of the radio's CRC status bit is made; if CRC failed, *rxPacketStatus* is set to CRC_ERROR; if CRC passed, *rxPacketStatus* is set to PKT_OK. In any event, Timer 2 will be reset by setting register T2 to FRAME_TIMEOUT_DEFAULT and the radio forced to the IDLE state. The radio's channel register (CHANNR) is then set to the next channel in the current band, variable *ActiveChIdx* incremented (modulo 4), and PLL calibration initiated.

If *rxPacketStatus* indicates that a timeout error occurred, "nulls' are copied into the *audioIn* buffer using DMA channel 3. Variable *lostpkts* is incremented and, if greater than 4, is reset to 0 and the *waitingforbeacon* flag is set to TRUE. If not (a time out error did not occur), variable *lostpkts* is reset. If *rxPacketStatus* equals PKT_OK, the 96 ADC samples are copied from the *SLVrxdada* buffer into the *audioIn* buffer using DMA channel 3. If a CRC error occurred, the *audioIn* buffer will be filled with "nulls". After waiting for PLL calibration to complete (radio state equals MARCSTATE_IDLE), the loop is restarted.

7 Conclusion

This document has shown how a microphone can be wireless interfaced to a USB audio device using the Texas Instruments CC1110 and CC1111 Low-Power SoC RF Transceivers and the TLV320AIC3104 audio Codec. The audio sample rate is 16 kHz and full 16 bit ADC resolution is maintained, suitable for voice quality applications.

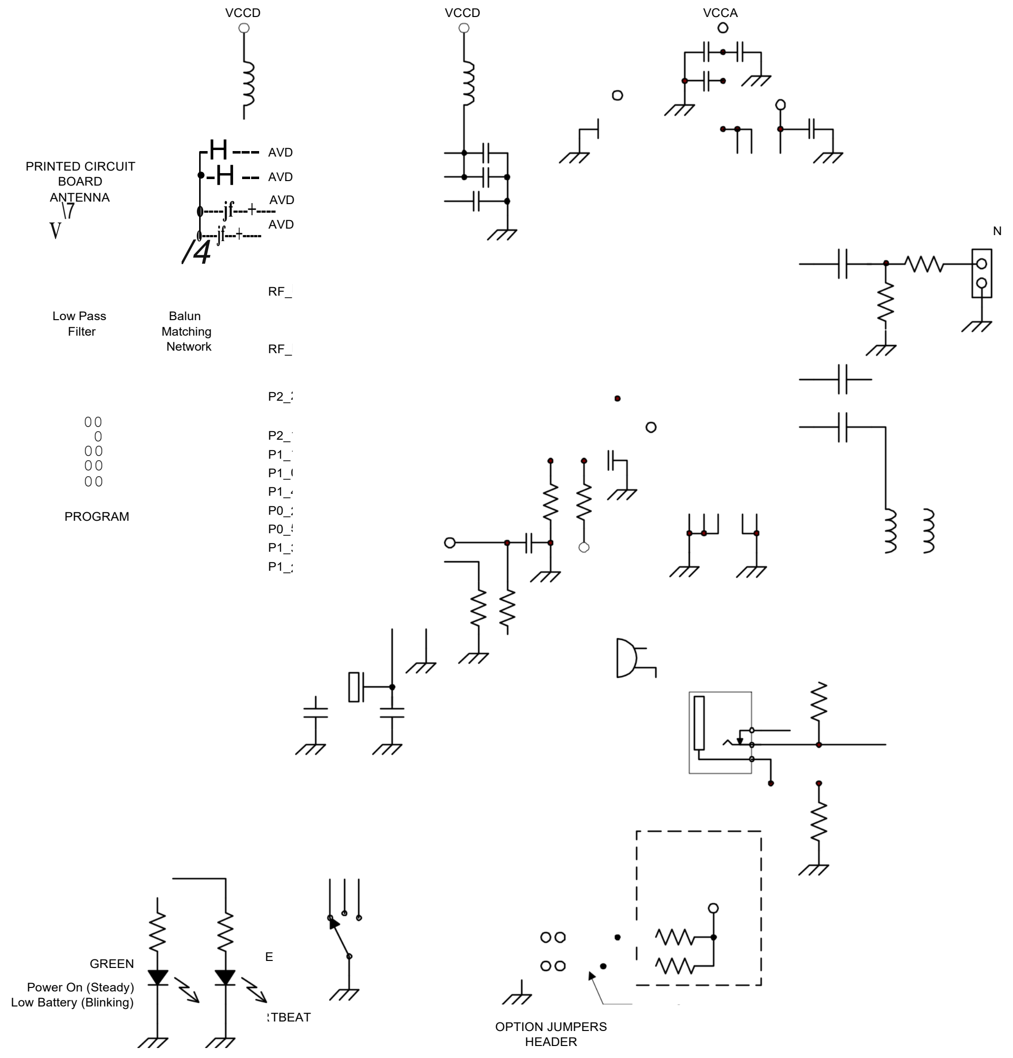


Figure 5. Microphone PCBA Block Diagram (1 of 2)

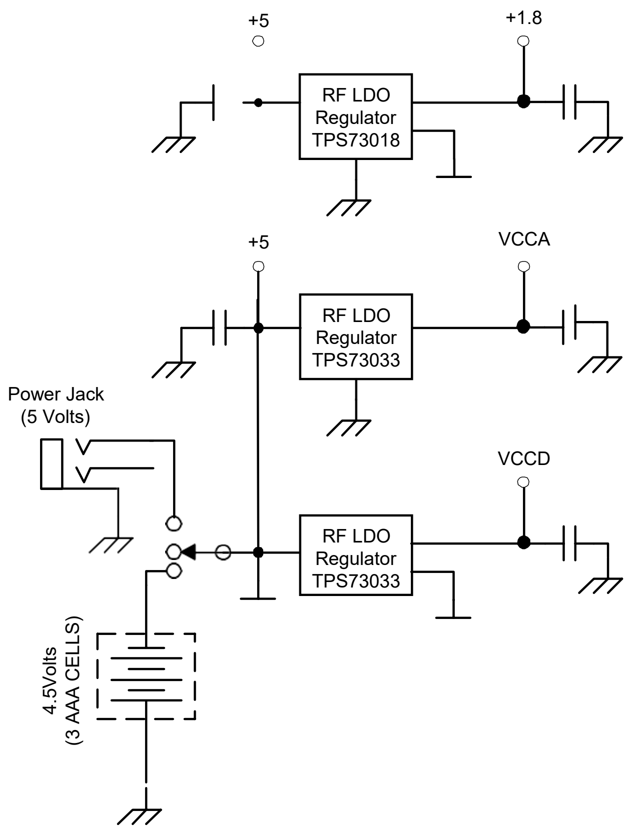


Figure 6. Microphone PCBA Block Diagram (2 of 2)

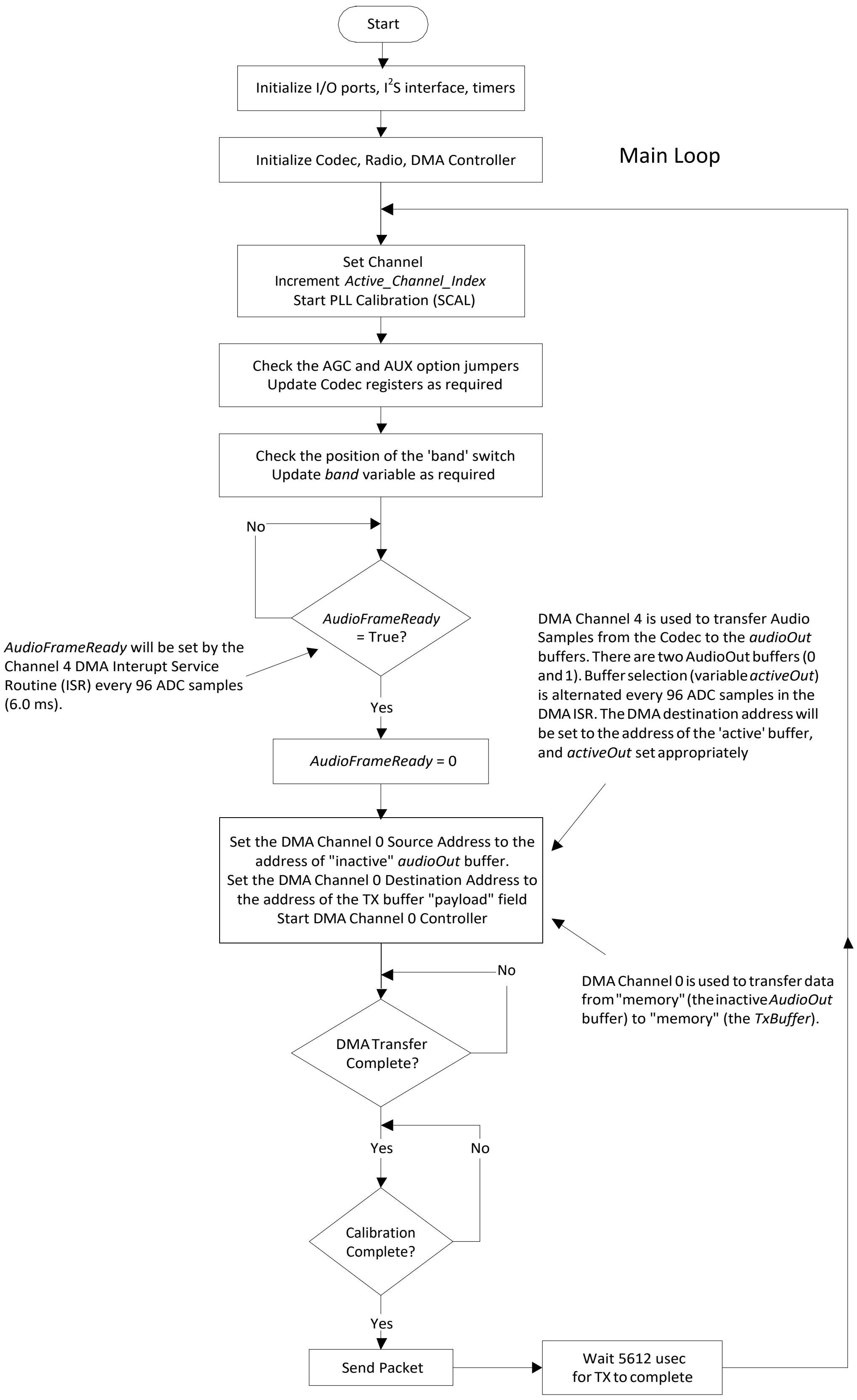

Start
Initialize I/O ports, I²S interface, timers
Initialize Codec, Radio, DMA Controller
Main Loop
Set Channel
Increment *Active_Channel_Index*
Start PLL Calibration (SCAL)
Check the AGC and AUX option jumpers
Update Codec registers as required
Check the position of the 'band' switch
Update *band* variable as required
No
AudioFrameReady = True?
AudioFrameReady will be set by the Channel 4 DMA Interupt Service Routine (ISR) every 96 ADC samples (6.0 ms).
DMA Channel 4 is used to transfer Audio Samples from the Codec to the *audioOut* buffers. There are two AudioOut buffers (0 and 1). Buffer selection (variable *activeOut*) is alternated every 96 ADC samples in the DMA ISR. The DMA destination address will be set to the address of the 'active' buffer, and *activeOut* set appropriately
Yes
AudioFrameReady = 0
Set the DMA Channel 0 Source Address to the address of "inactive" *audioOut* buffer.
Set the DMA Channel 0 Destination Address to the address of the TX buffer "payload" field
Start DMA Channel 0 Controller
No
DMA Channel 0 is used to transfer data from "memory" (the inactive *AudioOut* buffer) to "memory" (the *TxBuffer*).
DMA Transfer Complete?
Yes
No
Calibration Complete?
Yes
Send Packet
Wait 5612 usec for TX to complete

Figure 7. Microphone PCBA Main Program Flow Chart

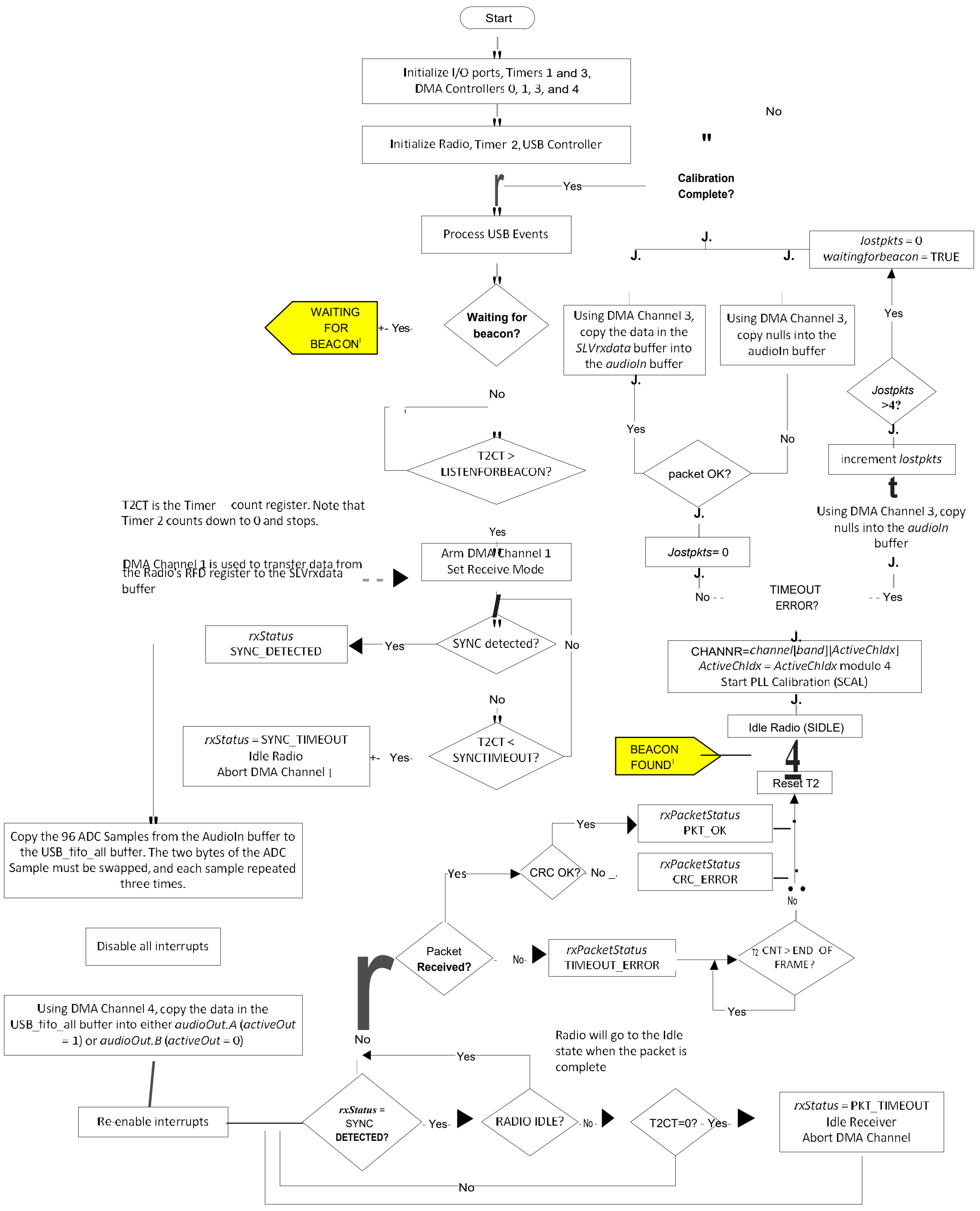
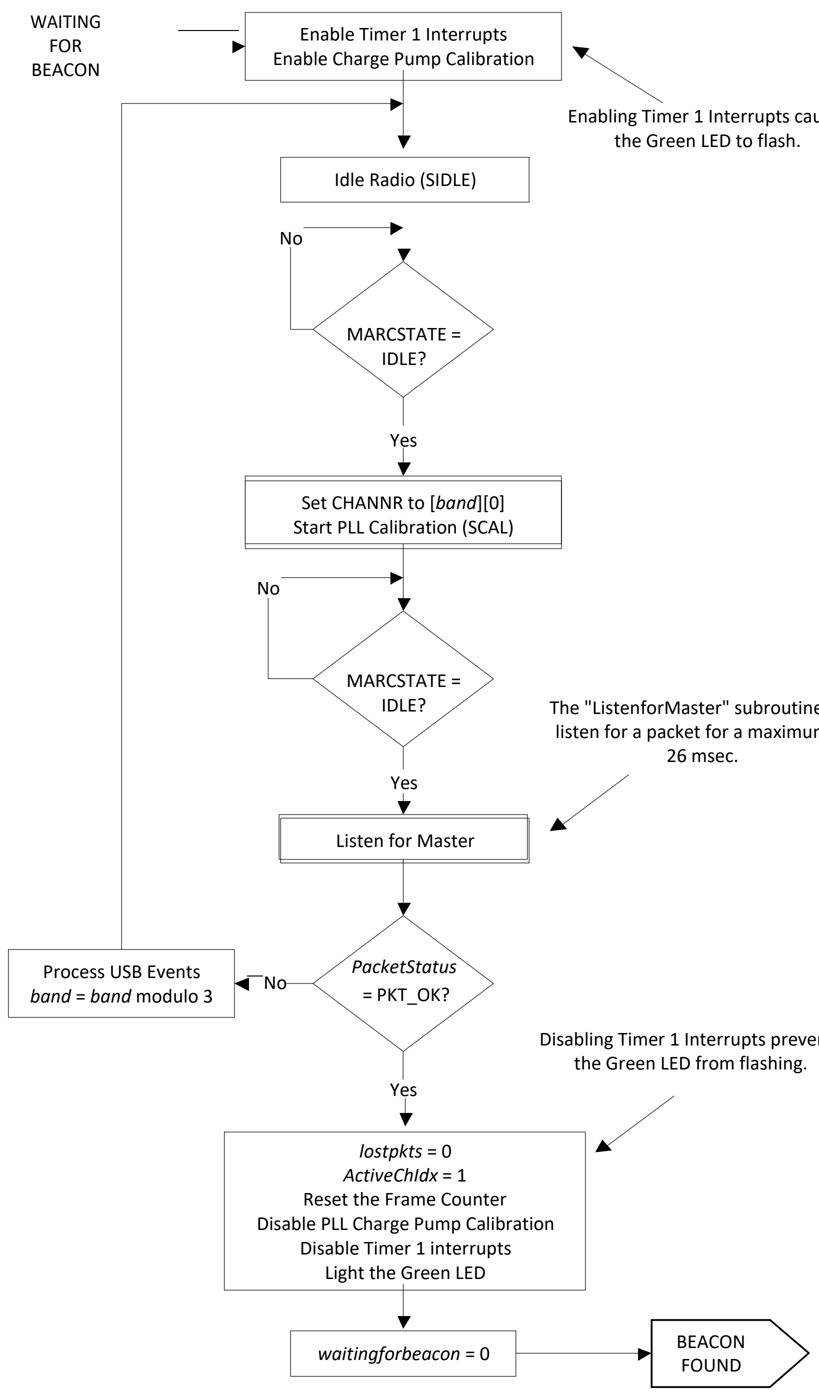

Start
Initialize I/O ports, Timers 1 and 3, DMA Controllers 0, 1, 3, and 4
Initialize Radio, Timer 2, USB Controller
Yes
No
Calibration Complete?
Process USB Events
lostpkts = 0
waitingforbeacon = TRUE
WAITING FOR BEACON[1]
Yes
Waiting for beacon?
Using DMA Channel 3, copy the data in the SLVrxdata buffer into the audioIn buffer
Using DMA Channel 3, copy nulls into the audioIn buffer
Yes
lostpkts >4?
No
T2CT > LISTENFORBEACON?
Yes
No
packet OK?
increment lostpkts
T2CT is the Timer count register. Note that Timer 2 counts down to 0 and stops.
Using DMA Channel 3, copy nulls into the audioIn buffer
Yes
Arm DMA Channel 1
Set Receive Mode
lostpkts= 0
DMA Channel 1 is used to transfer data from the Radio's RFD register to the SLVrxdata buffer
No
TIMEOUT ERROR?
Yes
rxStatus SYNC_DETECTED
Yes
SYNC detected?
No
CHANNR=channel[band][ActiveChIdx]
ActiveChIdx = ActiveChIdx modulo 4
Start PLL Calibration (SCAL)
No
Idle Radio (SIDLE)
rxStatus = SYNC_TIMEOUT
Idle Radio
Abort DMA Channel 1
Yes
T2CT < SYNCTIMEOUT?
BEACON FOUND[1]
Reset T2
Copy the 96 ADC Samples from the AudioIn buffer to the USB_fifo_all buffer. The two bytes of the ADC Sample must be swapped, and each sample repeated three times.
Yes
rxPacketStatus PKT_OK
Yes
CRC OK?
No
rxPacketStatus CRC_ERROR
No
Disable all interrupts
Packet Received?
No
rxPacketStatus TIMEOUT_ERROR
T2 CNT > END OF FRAME?
Yes
Using DMA Channel 4, copy the data in the USB_fifo_all buffer into either audioOut.A (activeOut = 1) or audioOut.B (activeOut = 0)
No
Radio will go to the Idle state when the packet is complete
Yes
Re-enable interrupts
rxStatus = SYNC DETECTED?
Yes
RADIO IDLE?
No
T2CT=0?
Yes
rxStatus = PKT_TIMEOUT
Idle Receiver
Abort DMA Channel
No

Figure 8. USB Dongle Main Program Flow Chart

[1] See Figure 9 for "Waiting for Beacon" code



WAITING FOR BEACON
Enable Timer 1 Interrupts
Enable Charge Pump Calibration
Enabling Timer 1 Interrupts causes the Green LED to flash.
Idle Radio (SIDLE)
No
MARCSTATE = IDLE?
Yes
Set CHANNR to [*band*][0]
Start PLL Calibration (SCAL)
No
MARCSTATE = IDLE?
Yes
The "ListenforMaster" subroutine will listen for a packet for a maximum of 26 msec.
Listen for Master
PacketStatus = PKT_OK?
No
Process USB Events
band = *band* modulo 3
Yes
Disabling Timer 1 Interrupts prevents the Green LED from flashing.
lostpkts = 0
ActiveChIdx = 1
Reset the Frame Counter
Disable PLL Charge Pump Calibration
Disable Timer 1 interrupts
Light the Green LED
waitingforbeacon = 0
BEACON FOUND

Figure 9. "Waiting for Beacon" Code

8 References

[1] CC1110\CC1111 True System-on-Chip Low Power RF Transceiver and 8051 MCU (swrs033.pdf)

[2] CC1111 USB Evaluation Module Kit 868/915 MHz (http://focus.ti.com/docs/toolsw/folders/print/cc1111emk868-915.html)

[3] CC1110EM 868 and 915 MHz Reference Design (swrr048.zip)

[4] CC1111 USB Evaluation Module Kit 868/915 MHz (http://focus.ti.com/docs/toolsw/folders/print/cc1111emk868-915.html)

[5] CC USB Firmware Library and Examples (swrc088.zip)

[6] SmartRF Studio (swrc176.zip)

[7] TLV320AIC3104 Low Power Stereo Audio Codec (slas510.pdf)

[8] TPS730xx Low-Noise, high PSRR, RF 200-ma, Low-Dropout Linear Regulators (sbvs054.pdf)