Application Note

Interfacing the C2000™ With an AFE03x: B-FSK Example

Vincent Rodriguez, Errol Leon, Kevin Allen

ABSTRACT

This application report provides information on how to interface a C2000 device with an AFE031, and send and receive data over a wired coupled interface using frequency shift keying. A demonstration of transmitting FSK information is discussed with tone frequencies in the 130-145 kHz range.

The target processor for the software is the TMS320F28379D, but the software can be ported to other devices. Project collateral and example code discussed in this document can be found in the latest C2000Ware release within the directory:

C:\ti\c2000\C2000Ware_x_xx_xx_xx\device_support\f2837xd\examples\cpu1

The example projects available include:

- boostxl_afe031_f28379d_pwmmode
- boostxl_afe031_f28379d_dacmode
- boostxl_afe031_f28379d_rx

A reference design for the BOOSTXL-AFE031 can be found at the following URL: http://www.ti.com/tool/TIDA-060001.

Table of Contents

List of Figures

List of Tables

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.
All trademarks are the property of their respective owners.

1 FSK Overview

Frequency Shift Keying (FSK) is a modulation scheme that utilizes discrete changes of frequency to transmit and receive digital data. One of the simplest subsections of this modulation scheme, and also the modulation used in this demo, is called Binary Frequency Shift Keying (BFSK).

In this scheme, the system is switching between two discrete frequencies: the Mark Frequency (“1”) and the Space Frequency (“0”). These frequencies correlate directly to the bit value of the transmitted data.

Figure 1-1 shows what this looks like in the time domain.

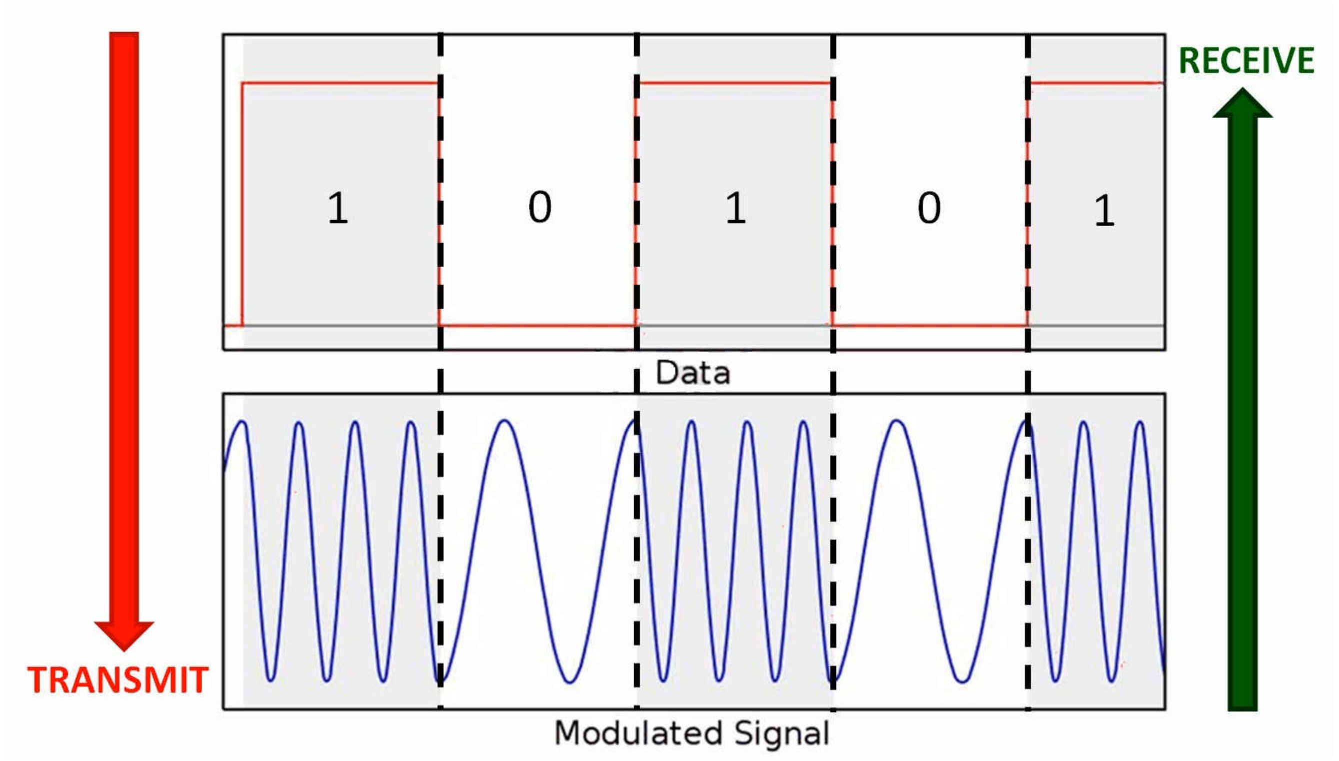


Figure 1-1. Binary FSK in the Time Domain

Figure 1-2 shows an example of a simplified FSK transmitter where the block consists of two oscillators with an internal clock as well as an input binary sequence to control the position of the switch.

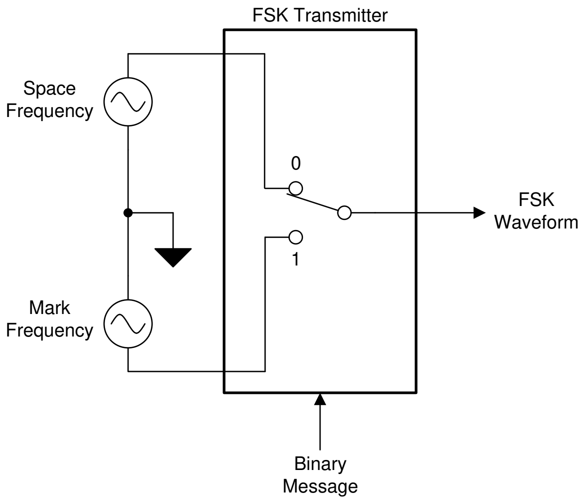


Figure 1-2. Transmitter Example

The two oscillators, producing a higher (space) and a lower (mark) frequency signals, are connected to a switch along with an internal clock. A clock is applied internally to both oscillators to avoid phase discontinuities of the output waveform during the transmission of the message. The binary input sequence is applied to choose the frequencies according to the binary input. In this case, binary "0" corresponds to the output of the space frequency and binary "1" corresponds to the output of the mark frequency.

Figure 1-3 shows an example of a simplified FSK receiver to convert a received signal back into the desired digital information.

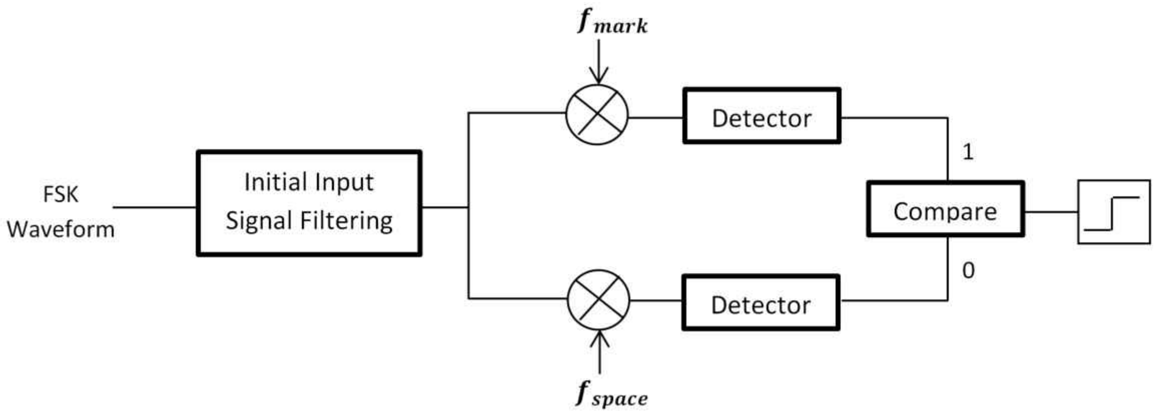


Figure 1-3. Receiver Example

A FSK waveform is initially filtered and then mixed with signals of the desired mark (f_{mark}) and space (f_{space}) frequencies. The output is run through a detector algorithm and the results are compared to decipher if the signal being received pertains to a mark, binary "1", or space, binary"0". Additional functionality is included to decipher received bits, based on the duration of the received mark or space signal, and handle the boundaries between consecutive bits.

This is a simple overview on how FSK works. The following sections discuss how this is implemented on a C2000 device.

2 Hardware Overview

The system created for the FSK transmitter is a combination of the following boards:

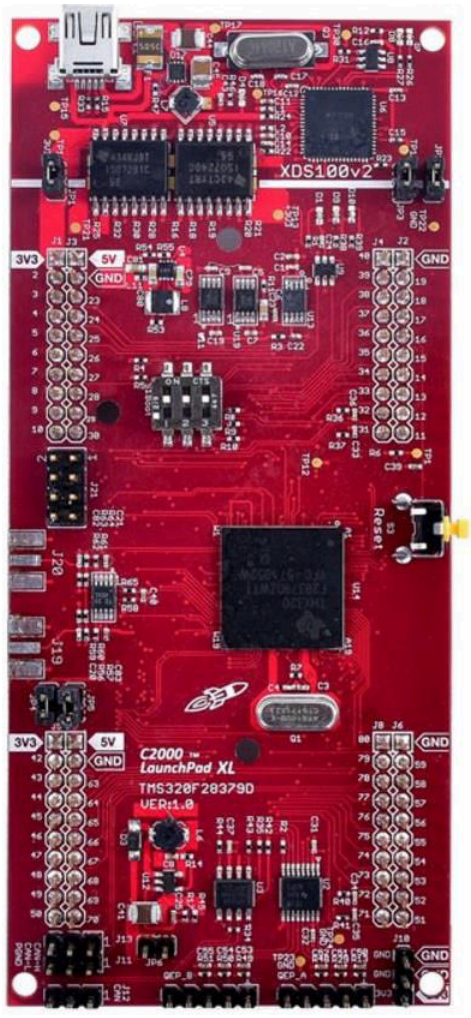


Figure 2-1. TMDS28379D LaunchPad

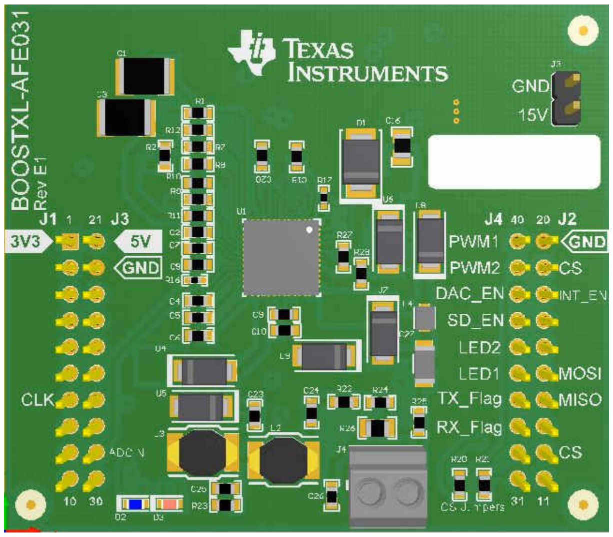


Figure 2-2. BoosterPack Test Board

The BoosterPack Test Board design files can be found in C2000Ware.The schematics are also included in Section 9.

2.1 Block Diagram

Figure 2-3 shows a block diagram of the system. In this example, not all of the connections have been utilized, but all are present on the BoosterPack for future development.

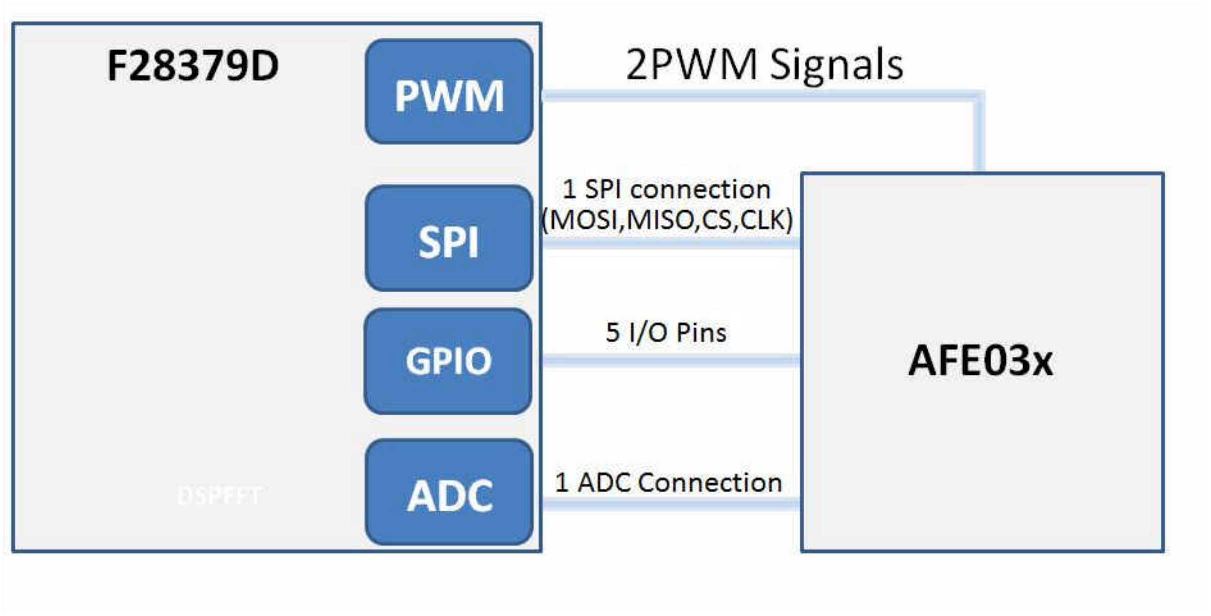


Figure 2-3. C2000 and AFE031 Block Diagram

The AFE has multiple internal registers that allow configuration of the internal components of the AFE chip, including filter selection, gain selection, and mode selections. These registers can be accessed using the SPI peripheral.

The AFE also has various GPIOs that allow the MCU to set the AFE into certain modes, as well as receive interrupts for critical events on the AFE. The ADC connection allows the MCU to receive or sample an input signal. The PWM signals provide a way to create an output for the AFE. Currently the AFE031 supports two modes of data transmission, PWM mode and DAC mode. An explanation and implementation of both of these modes can be found in Section 3.

2.2 Hardware Setup

There are a few things that need to be done so that the hardware can be debugged correctly. Power is supplied to the BOOSTXL-AFE031 test board via the 15 V jumper. The BoosterPack has a regulator that will supply power to the LaunchPad connected. Figure 2-4 shows a close up of the 15 V headers.

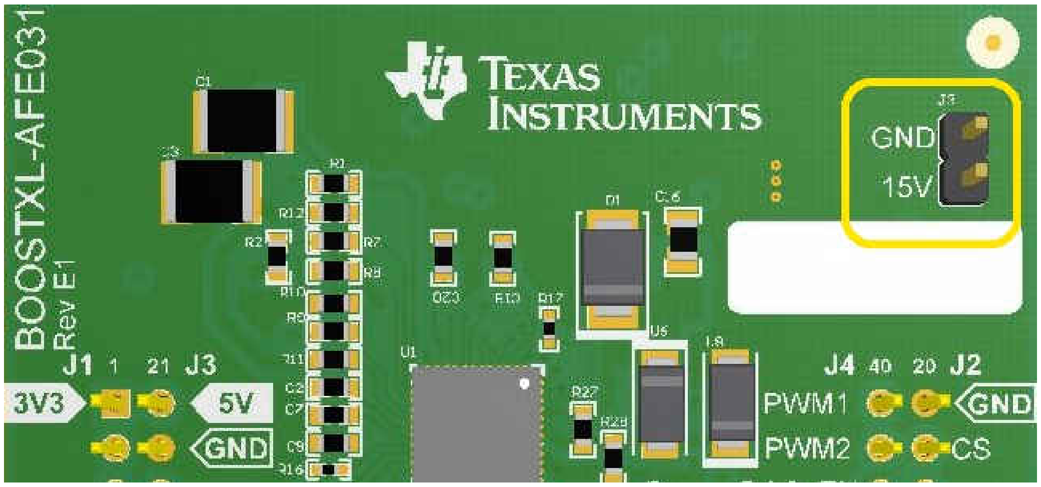


Figure 2-4. BOOSTXL-AFE031 Power Pins

The wire connected to the right input of the terminal block shown in Figure 2-5 will be the line that the transmitted output or received input FSK signal resides on, depending on whether the system is set up as a transmitter or receiver. The wire connected to the left input of the terminal block is the ground line. These lines can then be connected to some coupling circuitry or directly to another BOOSTXL-AFE031 terminal block for controlled testing, TX/RX to TX/RX and GND to GND.

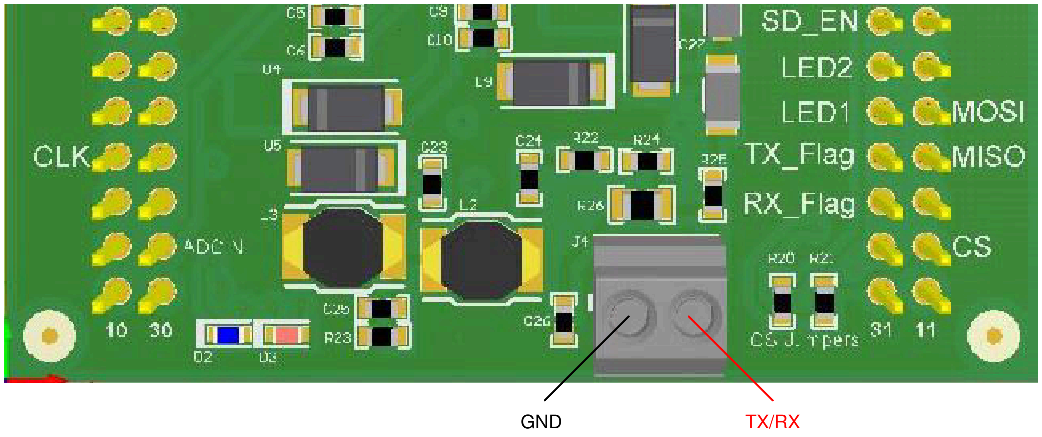


Figure 2-5. BOOSTXL-AFE031 Terminal Block Connections

Since the BoosterPack is supplying power to the launchPad, the PC USB interface needs to be isolated. Header jumpers JP1, JP2, and JP3 on the launchpad need to be removed to enable electrical isolation. Figure 2-6 is a close up of the location of the headers.

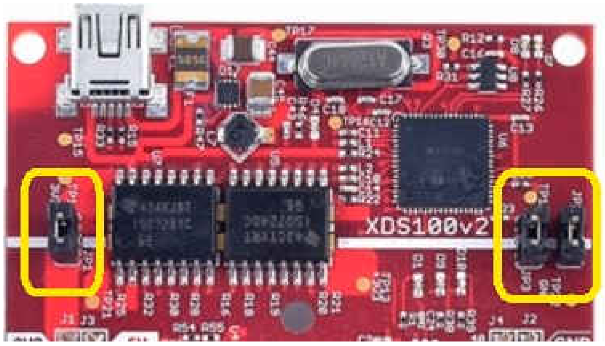


Figure 2-6. F28379D Jumper Configuration

When using the system as a FSK receiver, the LaunchPad needs 5 V to power the C2000 ADC. With the LaunchPad being isolated from the PC USB interface, the 5 V power rail needs to be generated by stepping up the supplied 3.3 V. Header JP6 on the LaunchPad needs to be added to enable the step-up regulator. Figure 2-7 is a close up of the location of the header.

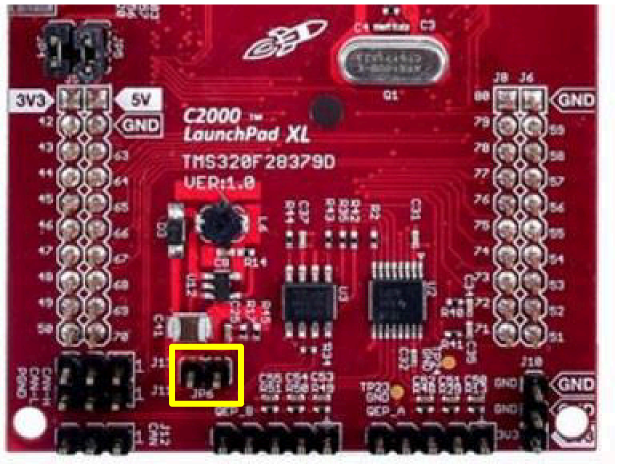


Figure 2-7. F28379D JP6 Location

When using the system as a receiver, make sure that the F28379D LaunchPad is version 2.0 or greater. Earlier versions (Ver 1.1 and 1.2) have an issue with the ADCIN pin that is connected to the BOOSTXL-AFE031. For more information, see the *LAUNCHXL-F28379D Overview* revisions section.

When using the transmitter solution in conjunction with the receiver solution, the complete system connection should look similar to what is shown in Figure 2-8.

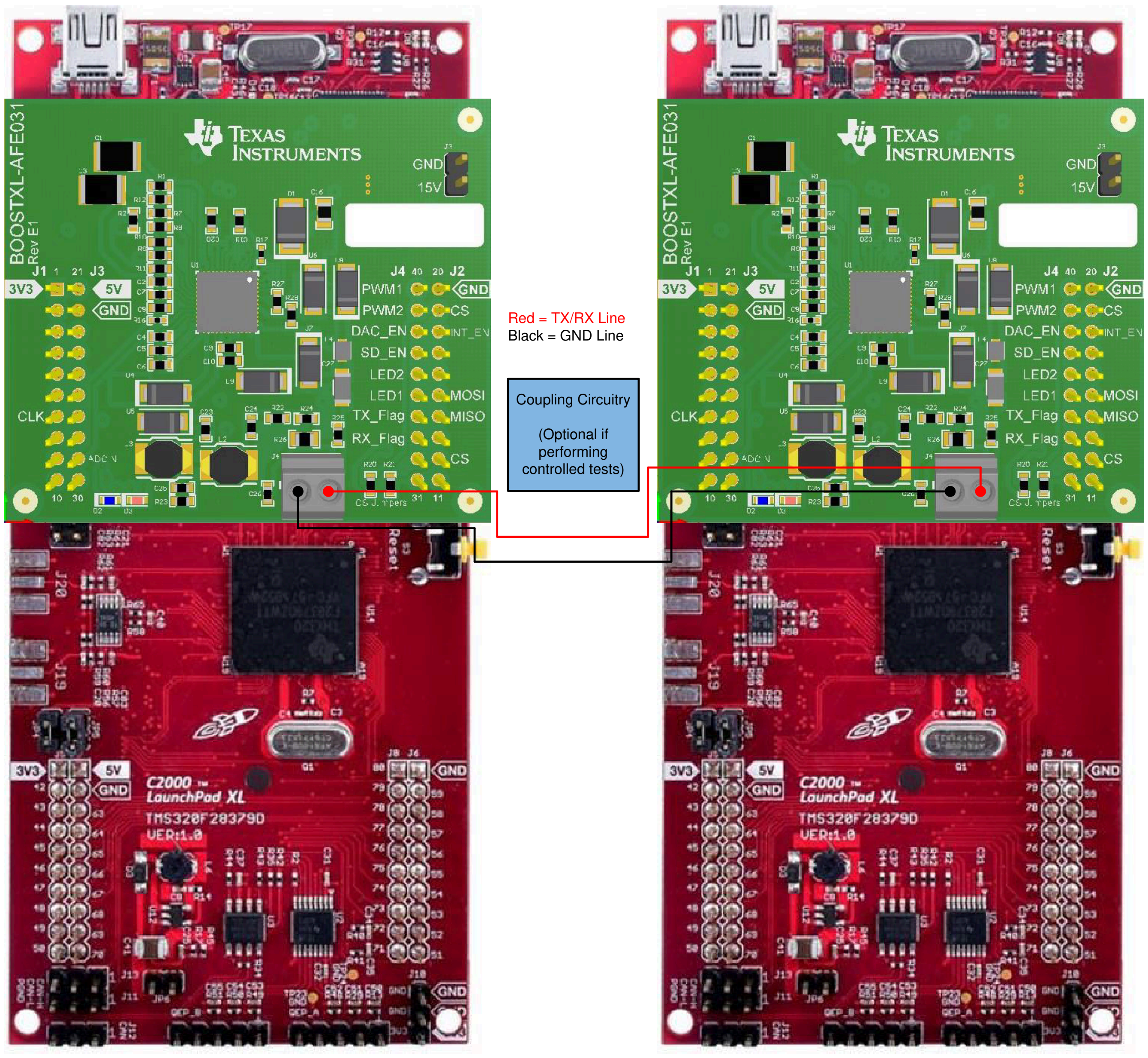


Figure 2-8. TX-RX Solution System

3 Interfacing With the AFE03x

When interfacing with the AFE03x as a transmitter, first decide which mode of operation to use for the AFE03x, DAC mode or PWM mode.

In DAC mode, the C2000 sends SPI data to update an internal DAC on the AFE031; the DAC value output is filtered and amplified. In this mode, a sine table is cycled through, and each point is sent to the AFE031 over SPI.

In PWM mode, the SPI connection is only used to configure the registers for the AFE. Two PWM signals are generated and supplied to the AFE031 at the desired frequency for FSK transmission. The AFE will add these signals together to create a waveform that creates less harmonics than a single PWM signal.

Note that these two modes require different hardware configurations, so be sure to choose one or the other. If using the BOOSTXL-AFE031 test board as a reference, the hardware has been designed so that both modes can be used by simply adding or removing components.

When interfacing with the AFE03x as a receiver, the SPI connection is only used to set the registers of the AFE, much like the PWM transmit mode. The receive path of the AFE is enabled to allow the input signal to reach the ADC input of the C2000. Along this path lies some filters and two programmable-gain amplifiers (PGA) to step up or step down the input signal before entering the C2000 ADC.

3.1 Configuring the AFE031

There are two main steps to configure the AFE device. First, ensure that the GPIOs are configured correctly. On the AFE device, two main GPIOs are set up: DAC Mode Select and System Shutdown. Both of these pins are pulled low. All other GPIOs connected can also be pulled low.

The next step is to configure the device over SPI. In the software example, the *HAL_afe031Init()* function configures the AFE031 and is defined within AFE03x_Config.c.

The steps below show the sequence the function follows to correctly configure the device:

1. Configure GPIOs.
   a. Setup SD and DAC pins on the AFE device. Both these pins need to be brought low.
   b. Software example function: *HAL_afe031_cfgGpio();*
2. Configure SPI.
   a. Configure SPI module for 16-bit characters. For more information regarding SPI requirements, see the *AFE031 Powerline Communications Analog Front-End Data Sheet*.
   b. Software example function: *HAL_spi_cfg();*
3. Perform a soft reset on the AFE device.
   a. Write 0x14 to the reset register.
   b. Software example function: *HAL_afe031_softReset();*
4. Enable the Bias.
   a. Write 0x03 to the enable2 register.
   b. Software example function: HAL_afe031_biasEnable();
5. Select a Frequency Band.
   a. Write either 1 or 0 to the CA_CBCD bit inside the Control1 Register. If a 1 is written, the frequency response of the TX and RX filters will be configured to CENELEC B,C,D. If a 0 is written, then CENELEC A will be configured.
   b. Software example function: *HAL_afe031_bandSelect(1);*
6. Clear all interrupts
   a. Write 0x00 to the control2 register.
   b. Software example function: *HAL_afe031_clrAllInt();*
7. Configure all interrupts.
   a. Interrupts can be configured by wrtiting to the control2 register. The software example function currently only enables the T_flag that indicates thermal overload.
   b. Software example function: *HAL_afe031_cfgInt();*
8. Enable zc.
   a. Write a 1 to the ZC bit in the enable2 register.

   b. Software example function: *HAL_afe031_zcEnable();*
9. Write TX Gain - If Transmitting
   a. Write either a 0, 1, 2, or 3 to the TXG bits in the gain select register. The gains are as follows:
      i. 0 =0.25 V/V
      ii. 1 = 0.5 V/V
      iii. 2 = 0.707 V/V
      iv. 3 = 1 V/V
   b. Software example function: *HAL_afe031_writeTxGain(UINT16 gain);*
      i. Note that the software example function uses an array to write the 0-3. Thus when using the function, the input parameter corresponds to the indice of the desired *HAL_afe031_txGainLut* array element. For example passing 0 to the function corresponds to a gain of 0.25 V/V.
10. Write RX Gain - If Receiving.
    a. Write 0-15 to the RXG bits in the gain select register. The gains range from 0.25 V/V, when RXG is set to 0x0, to 128 V/V, when RXG is set to 0xF.
       i. Refer to the Table 5-1 for specific gains.
    b. Software example function: *HAL_afe031_writeRxGain(UINT16 gain);*
       i. Note that the software example function uses an array to write the 0-15. Thus, when using the function, the input parameter corresponds to the indice of the desired *HAL_afe031_rxGainLut* array element. The included array does not have values for all RX gain configurations.

Once the above is complete, begin the final few configurations needed based on the transmit or receive implementation being used.

4 Transmit Path

Now that the AFE is initialized correctly, enable the system to transmit based on which transmit method is chosen: PWM Mode and DAC Mode.

4.1 FSK Example Specifications

Table 4-1 describes the parameters around the FSK signal that is generated in the software example.

The end goal is to send the complete packet, wait the designated wait period, and repeat.

Table 4-1. FSK Specifications

Symbol	TX Specification	Min	Nom	Max	Unit	Comment
W1	Logic 1 code word	{{-1,-1,-1,+1,+1,+1,-1,+1,+1,-1,+1}}				+1 = Mark, -1 = Space
W0	Logic 0 code word	{{+1,+1,+1,-1,-1,-1,+1,-1,-1,+1,-1}}				+1 = Mark, -1 = Space
packet_1	Complete Packet	ABC = [W1,W1,W1]				
packet_0	Complete Packet	ABC = [W0,W0,W0]				
Fm	Mark Frequency	131.236875	131.25	131.263125	kHz	
Fs	Space Frequency	143.735625	143.75	143.764375	kHz	
Ts	Bit Period	5.119488	5.12	5.120512	ms	
Tt	Tx Period	168.943104	168.96	168.976896	ms	3 words
Tq	Quiet Period	901.029888	901.12	901.210112	ms	16 words
Tc	Cycle Period	1069.972992	1070.08	1070.187008	ms	19 words

4.2 PWM Mode

In PWM mode, the C2000 F28379D generates two symmetric PWM signals that go directly into the AFE device. The two symmetric PWM signals are 66% and 33% duty cycle. These signals are added together inside the AFE device and create a waveform that has the least amount of noise. Figure 4-1 shows how this addition works.

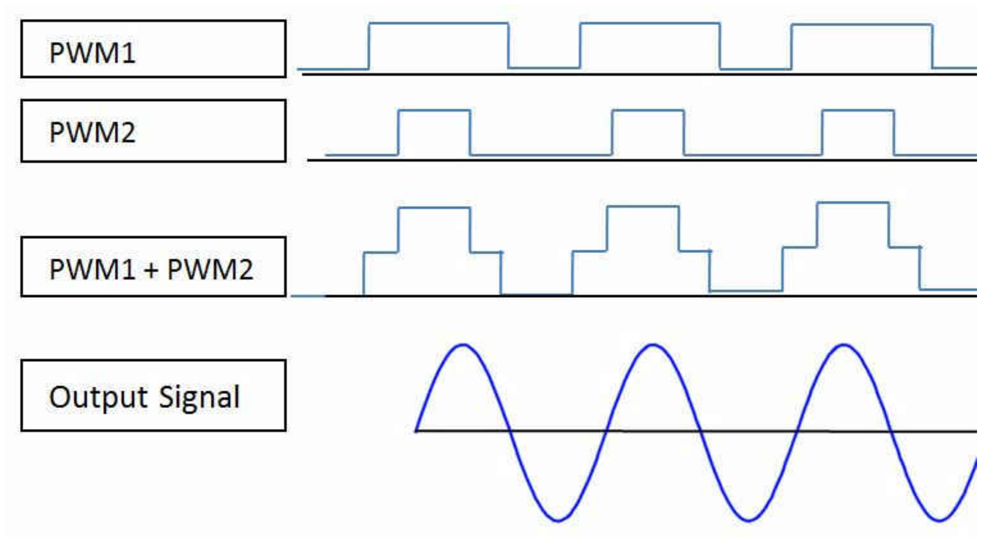


Figure 4-1. PWM Addition

The path of the PWM signals is shown in Figure 4-2. The PWM signals go into the low-pass filter internal to the AFE030/1 device, and are added together to create the above PWM1+PWM2 waveform.

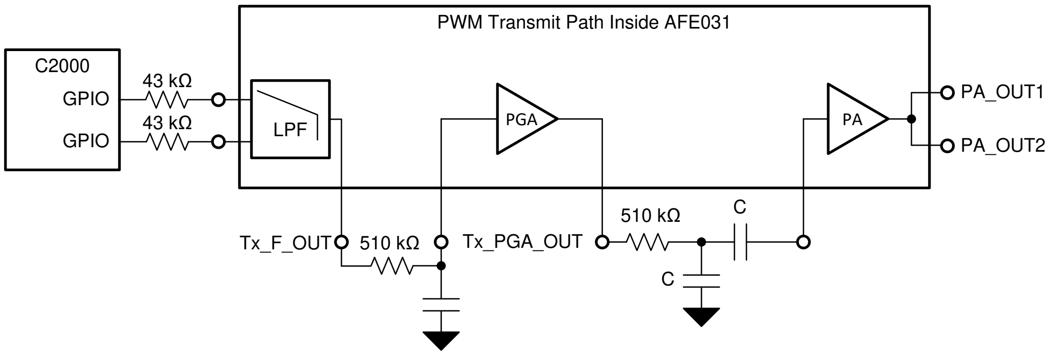


Figure 4-2. PWM Transmit Path

The gains witnessed at certain frequencies at the output of the internal TX low-pass filter are shown in Figure 4-3.

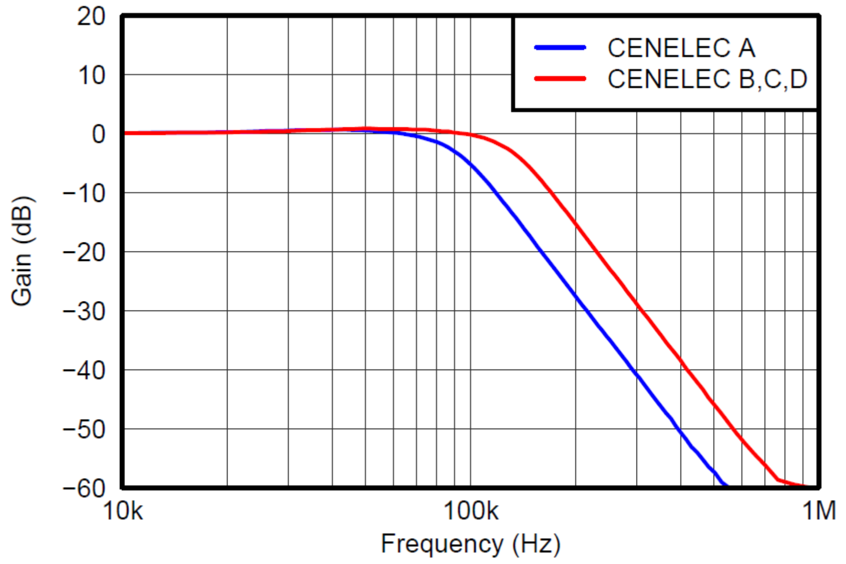


Figure 4-3. TX Filter Gain vs. Frequency

Next, the signal goes through a low-pass filter, PGA, another low-pass filter, and finally out of the PA. The external low-pass filters can be tuned to filter the desired frequencies shown in Table 4-2.

Table 4-2. External R and C Values to Increase Filter Response in PWM Applications

Frequency Band	R (Ω)	C (nF)
SFSK: 63 kHz, 74 kHz	510	2.7
CENELEC A	510	1.5
CENELEC B,C,D	510	1

4.2.1 Software Implementation

Example program referenced: boostxl_afe031_f28379d_pwmmode

To enable PWM mode in software, the following flow needs to be completed:

- Enable PWM transmit mode
  - To enable the PWM transmit mode, set the TX and PA bits in the enable register to 1, and set the DAC bit to 0.
  - Software example function: *HAL_afe031_txPWMEnable();*
- Disable the DAC transmit mode

Two PWM sources are used to create the two PWM signals: one PWM source is used to set the frequency of the two outputted signals and the other controls the bit rate for the sent data. In the software example, PWM2 is used to control the bit rate and generate an interrupt to determine the frequency that needs to be outputted.

In the software example, the PWM2 interrupt handles all of the FSK protocol requirements. The implemented protocol is a repeatable pattern, which allows the software to be based on a cycle count. One cycle count is the time period for one bit. In this implementation, 33 bits (11 bits per word, three words) are sent. During each cycle, it checks the value of the next bit and the PWM frequency changes to either the mark or space frequencies. After 33 cycles, the system stops sending PWM signals and enters the quiet mode. After 209 cycles, the cycle count is reset and the software starts sending the packet again.With the FSK transmission being handled by the PWM2 interrupt, the CPU's main function is free to be used for other applications. By default the software example will transmit a packet_1 referenced in Table 4-1, but this can be changed to a packet_0 by setting the packet_to_send variable to a zero.

4.2.2 Testing Results

Figure 4-4 and Figure 4-5 show that the system created both the mark and space frequencies. Also, Figure 4-6 shows the complete packet being sent out of the system.

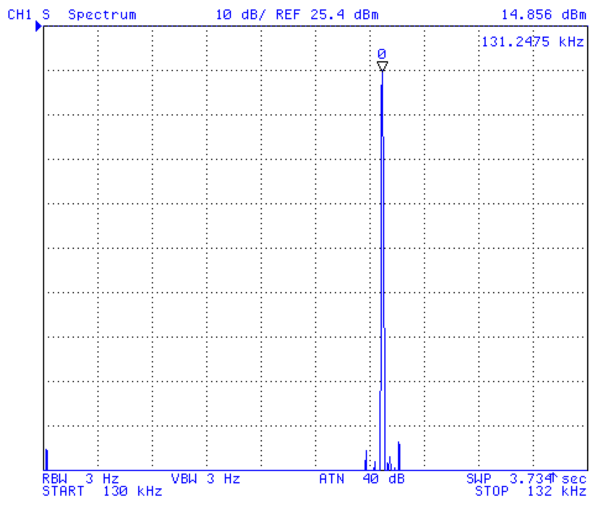


Figure 4-4. PWM Mode Mark Frequency, 131.25 kHz

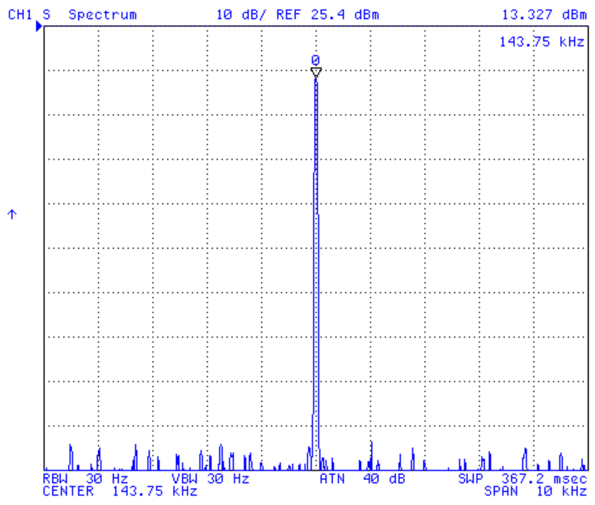


Figure 4-5. PWM Mode Space Frequency, 143.75 kHz

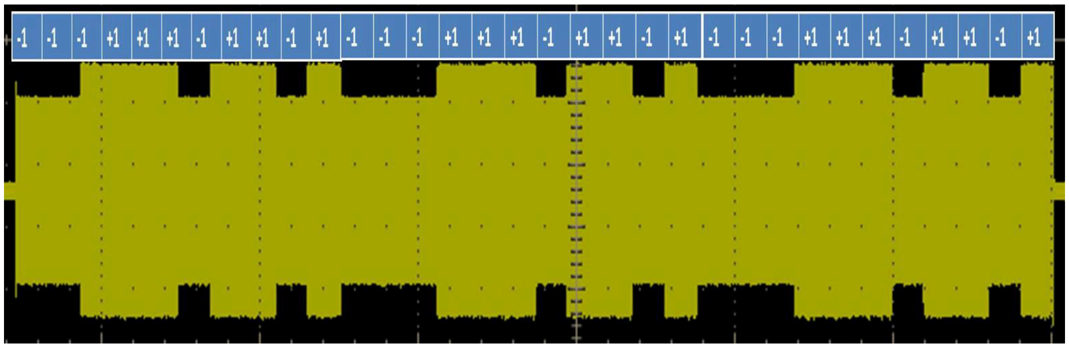


Figure 4-6. Full Packet Transmission Waveform

4.2.3 HRPWM vs. EPWM

Unique to the C2000 device is the ability to use High Resolution PWM (HRPWM). HRPWM enables increased resolution for both the duty cycle and period of the PWM signals. In this example, HRPWM is used to generate both the mark and the space frequencies.

The HRPWM is based on micro edge positioned (MEP) technology. MEP logic is capable of positioning an edge very finely by sub-dividing one coarse system clock of a conventional PWM generator. The time step accuracy is on the order of 150 ps.

Table 4-3 shows the resolutions possible with and without HRPWM.

Table 4-3. Resolution for PWM and HRPWM

PWM Frequency	Regular Resolution (PWM) 100 MHz EPWMCLK		High Resolution (HRPWM)	
(kHz)	Bits	%	Bits	%
20	12.3	0.02	18.1	0.000
50	11	0.05	16.8	0.001
100	10	0.1	15.8	0.002
150	9.4	0.15	15.2	0.003
200	9	0.2	14.8	0.004
250	8.6	0.25	14.4	0.005
500	7.6	0.5	13.4	0.009
1000	6.6	1	12.4	0.018
1500	6.1	1.5	11.9	0.027
2000	5.6	2	11.4	0.036

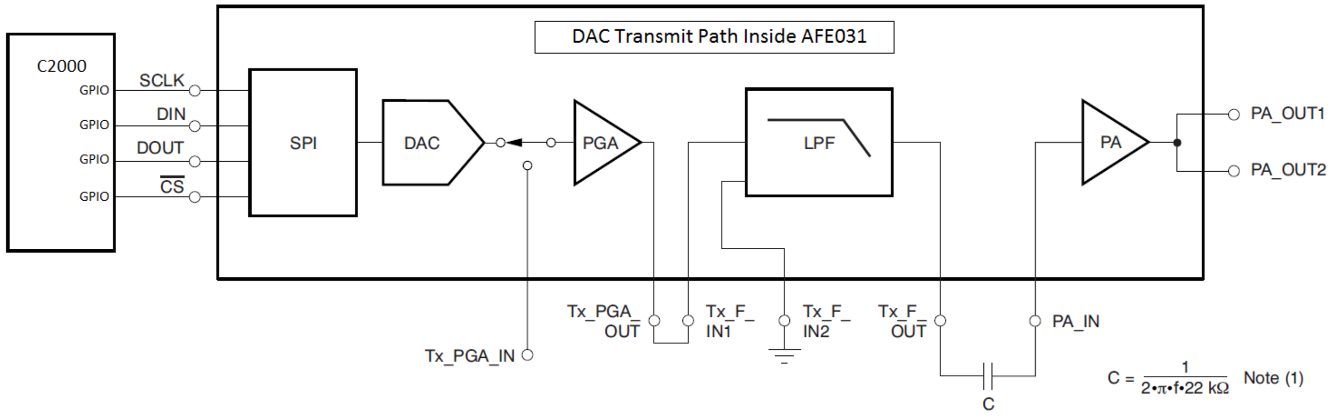
For example, the mark frequency was generated at 131.25 kHz signal. Using PWM, only a 131.2 kHz or a 131.3 kHz signal can be generated. This limitation is due to the resolution that is available for the PWM module. If HRPWM is added, a 131.250 kHz signal can be effectively generated.

If this amount of accuracy is not necessary or the desired frequency can be reached with normal PWM, HRPWM is not required. HRPWM is an add-on to PWM. To disable this in the software example, remove the code associated with it inside the main.c file.

4.3 DAC Mode

This section describes one way to create a DAC mode FSK transmitter. DAC Mode is very similar to the PWM Mode, in that PWM interrupts are used to accomplish FSK transmission.

The transmit path utilized when using DAC mode is shown in Figure 4-7.



(1) For capacitor value *C*, *f* is the desired lower cutoff frequency and 22 kΩ is the PA input resistance.

Figure 4-7. DAC Transmit Path

In DAC mode, the software sends data over SPI to the internal DAC to set the output value. To accomplish sending a single tone, or sine wave, a ramp of DAC values at a given frequency are sent. Using a PWM interrupt occurring every period, send the updated DAC value at a frequency determined by the PWM period. The number of points in the sine table and desired frequency will determine the frequency of the PWM signal. For example, if given a 100 kHz sine wave that has ten points, all ten points need to be sent within 100 kHz. This means that the PWM has to generate an interrupt that follows the following formula:

PWM Frequency = Points of Sine Table * Desired Frequency

Figure 4-8 shows what this looks like in the time domain.

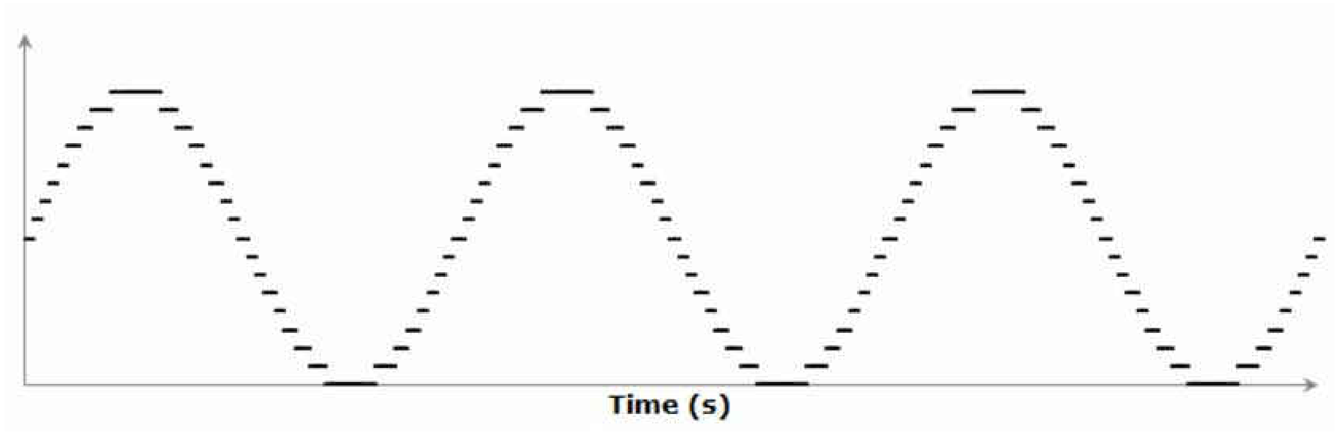


Figure 4-8. DAC Sine Wave Ramp

4.3.1 Software Implementation

Example program referenced: boostxl_afe031_f28379d_dacmode

To enable DAC mode in software, the following flow needs to be completed

- Enable DAC transmit mode internal to AFE.
  - To enable the PWM transmit mode, set the TX and PA bits in the enable register to 1, and set the DAC bit to 0.
  - Software example function: *HAL_afe031_txDACEnable();*
- Enable the DAC transmit mode with GPIO toggle and Configure word length for SPI
  - Set the GPIO connected the the DAC pin to 1.
  - set word length to 10 bytes for SPI communication.
  - Software example function: *HAL_afe031_dacEnable();*

Sending information in DAC mode, can be accomplished very similarly to how the PWM mode operates. One PWM source is used to set the DAC mode value to the correct value of the sine ramp that gets sent out. The second PWM is used for bit rate to generate an interrupt and determine what frequency needs to be outputted.

A problem does arise if attempting to reach vary precise frequencies with this implementation. The problem can be looked at with Equation 1.

Number of steps in Sine Table = (Frequency of Interrupt) / (Frequency of Desired Signals) (1)

If trying to generate 131.25 kHz, then only two variables can change. One approach is to set the number of steps in the sine table. For example, if there are 10 steps in the sine table the Frequency of interrupt is:

Frequency of Interrupt = (Number of Steps in Sine Table) * (Frequency of Desired Signal)

Frequency of Interrupt = 1.3125 MHz

With a 200 MHz clock,even if an interrupt occurs at either 152 or 153 CPU cycles, the interrupt frequencies would be 1.31579 MHz and 1.30719 MHz respectively. These frequencies do not fall within spec for the frequency tolerances in Table 4-1. This means setting the step size cannot be the correct implementation for precise frequency generation.

The other way to think of this is to set the frequency of interrupt. Let's set the interrupt for 1 MHz, something that is possible to generate. The number of steps in the sine wave would then be:

Number of steps in Sine Table = (1 MHz) / (131.25 kHz)

Number of Steps in sine Table = 7.61905

Utlizing the floating-point capability of the F28379D, the processor can keep track of that remainder, and now the accuracy depends on the sine table, not the 1 MHz clock. The step size can be found using the following formula:

Step Size = (Points in Sine Table)/ (Number of steps in Sine Table)

With a 4096 sine table, and continuing with the previous example, this gives 537.6 as the step size. This means in every interrupt, the sine table will step another 537.6. Since the program is sorting through an array, this number will be rounded off to 537, but as this number gets added and the program shifts through the sine table, the next step will vary off the decimal step size. An example interrupt routine is shown below:

- //Transmit next data point in sine wave. Also convert float SinePosition to unsigned int
  - Uint16 temp = sinePosition;
  - HAL_spi_xmt((sineTable[temp]));
- //Calculate next step
  - sinePosition += sineStep; }
- //Check for overflow
  - if(sinePosition > 4095)
  - { sinePosition -= 4095; }

One aspect to note is that in this implementation, a 1 MHz Interrupt was created, which only moves data from one memory address to another. A way to make this less CPU intensive is to utilize the C2000's DMA (Direct Memory Access). The DMA peripheral moves data from one memory address to another based on a trigger event. Pre-fill two buffers with the correct data that is chosen to send out over SPI, and use the DMA to switch between the two. When switching from one buffer to the other, the older buffer will refill with the values.

In the DAC Mode software, a 1 MHz PWM signal is utilized to generate a DMA event that moves data from one of these buffers to the SPI TX Buffer. Since the older buffer needs to be re-filled after each use, the DMA will trigger an event each time it finishes reading an entire buffer. During this interrupt, the buffers are switched, and the old one is re-filled. By following this implementation, CPU utilization vs. memory tradeoffs can be weighed and the buffers can be sized accordingly. If extra memory is available, using a bigger buffer will reduce the CPU overhead. If not a lot of memory is available, then the smaller the buffer size increases the CPU overhead.

Exactly like the PWM mode software implementation, the PWM2 interrupt handles all of the FSK protocol needs. The protocol being implemented is a repeatable pattern, which allows the software to be based on a cycle count. One cycle count is the time period for one bit. In this implementation, 33 bits (11 bits per word, three words) are being sent. During each cycle, a check to see the value of the next bit and the step size changes to allow the sine table to be sent out at the mark or space frequencies. After 33 cycles, the system stops sending DAC values and enters the quiet mode. After 209 cycles, the cycle count is reset and the software starts sending the packet again.With the FSK transmission being handled by the PWM2 interrupt, the CPU's main function is free to be used for other applications. By default the software example will transmit a packet_1 referenced in Table 4-1, but this can be changed to a packet_0 by setting the packet_to_send variable to a zero.

4.3.2 Testing Results

Figure 4-9 and Figure 4-10 show that the system created both the mark and space frequencies. Also, Figure 4-11 shows the complete packet being sent out of the system.

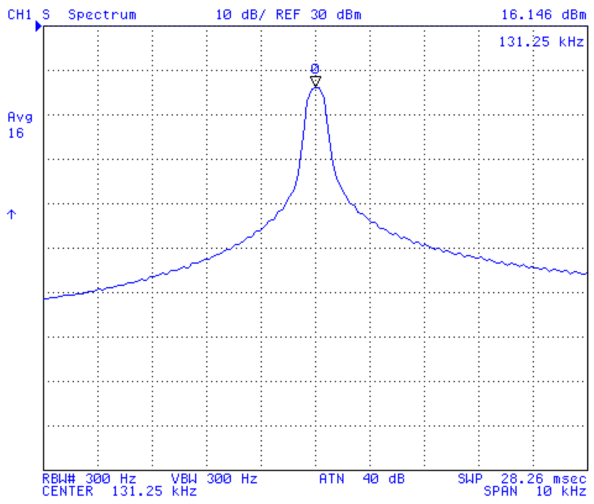


Figure 4-9. DAC Mode Mark Frequency Spectrum Analyzer

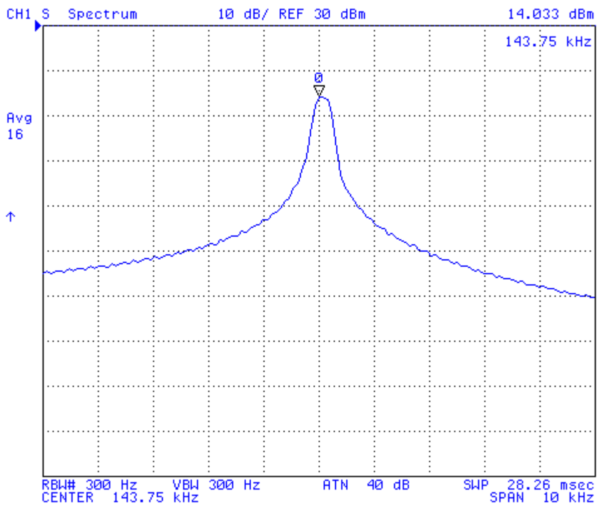


Figure 4-10. DAC Mode Space Frequency Spectrum Analyzer

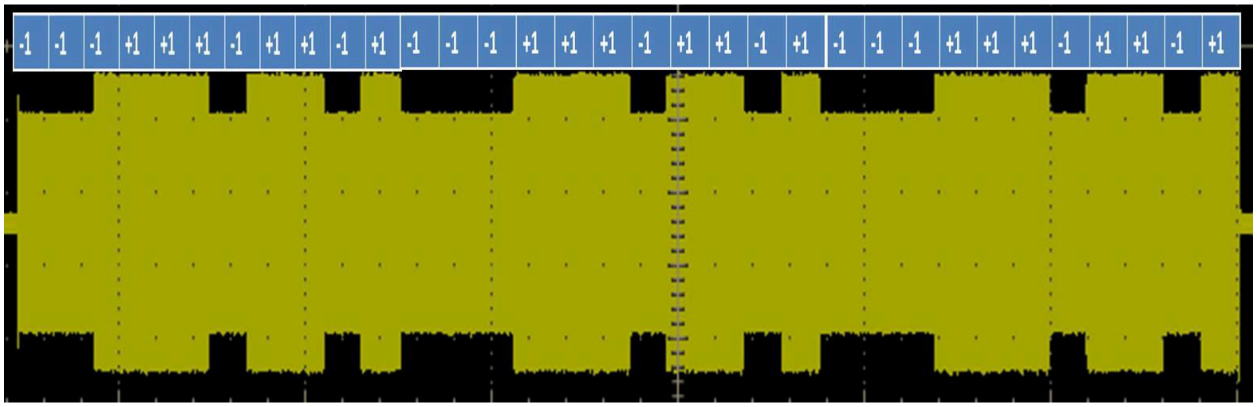


Figure 4-11. Full Packet Transmission Waveform

4.3.3 OFDM Ability

Although not implemented, the ability to do OFDM Frequency modulation is possible using the DAC mode of the AFE device. Being able to select the exact DAC value at every point gives the ability to combine frequency components on the MCU side, and set the DAC to the desired value.

An example of combining two frequencies is shown in Figure 4-12 through Figure 4-15, which showcases how this can be accomplished. The two signals are 50 Hz and 100 Hz sine wave. To send these out independently, each point is discretely sent at the correct time interval to create the sine wave.

To send both frequencies out at the same time, the frequencies would need to be combined first. The result is shown in Figure 4-14. To send out each point in this waveform, the system would be sending two frequencies out at once. An FFT is performed on the data to see the two discrete frequencies.

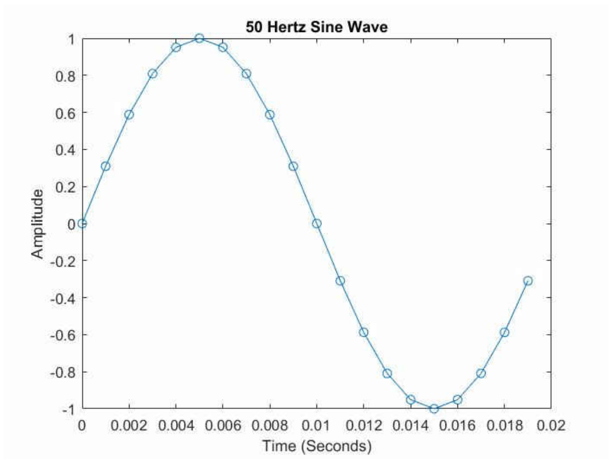


Figure 4-12. 60 Hz Sine Wave

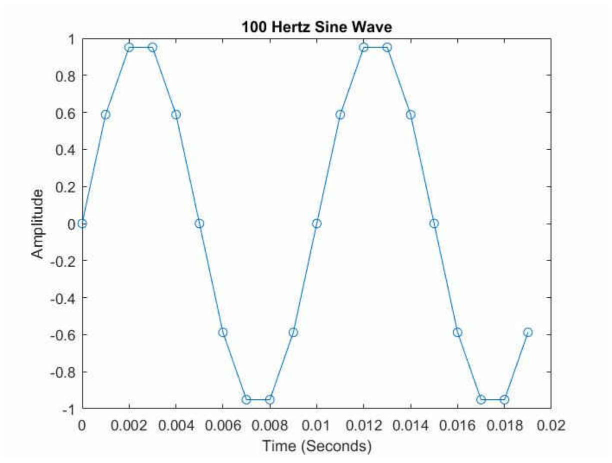


Figure 4-13. 100 Hz Sine Wave

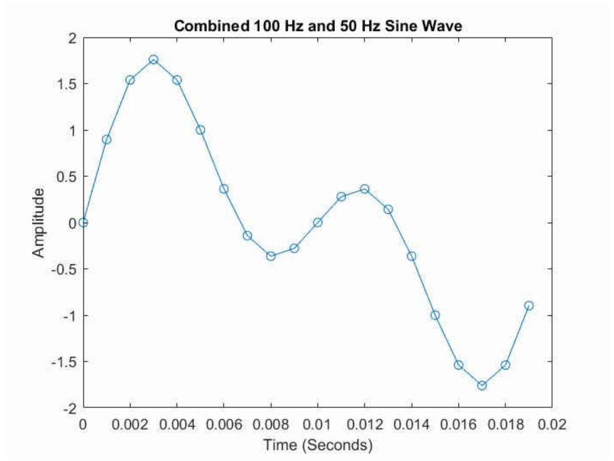


Figure 4-14. 60 + 100 Hz Sine Wave

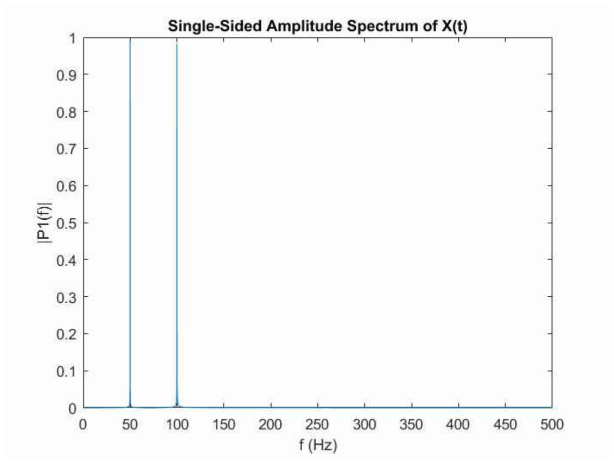


Figure 4-15. FFT of Combined Signals

4.4 Porting TX to LAUNCHXL-F280049C

Below are the steps taken to port boostxl_afe031_f28379d_pwmmode and boostxl_afe031_f28379d_dacmode from the LAUNCHXL-F28379D to the LAUNCHXL-F280049C after importing the projects into Code Composer Studio™.

1. Change the C2000Ware includes search path from f2837xd to f28004x.
   a. Right click the project --> select Properties --> expand Resource --> select Linked Resources.
   b. Select F28379D_DEVICE_SUPPORT_ROOT and click Edit.
   c. Change the name from F28379D_DEVICE_SUPPORT_ROOT to F28004X_DEVICE_SUPPORT_ROOT.
   d. Change the directory location to be \device_support\f28004x instead of \device_support\f2837xd.
   e. In Properties, expand CCS Build --> expand C2000 Compiler --> select Include Options.
   f. Alter include paths that have F28379D_DEVICE_SUPPORT_ROOT to F28004X_DEVICE_SUPPORT_ROOT.
   g. Select Apply and Close.
2. Replace all of the F2837xD source files within the project with f28004x source files.
   a. f28004x source files found within the C2000Ware directory C:\ti\c2000\C2000Ware_<version>\device_support\f28004x\common\source
3. Replace the 2837x linker command files with 28004x linker command files.
   a. Generic flash and RAM 28004x linker command files can be found in the C2000Ware directory C:\ti\c2000\C2000Ware_<version>\device_support\f28004x\common\cmd.
4. Alter the GPIO configurations for LED control within boostxl_afe031_f28379d_main.c
   a. Alter the GPIOs configured within the AFE_InitGpio function.

```
    //GPIO-23 - LaunchPad RED LED
    GPIO_SetupPinMux(23, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(23, GPIO_OUTPUT, GPIO_PUSHPULL);
    // GPIO-34 - LaunchPad GREEN LED
    GPIO_SetupPinMux(34, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(34, GPIO_OUTPUT, GPIO_PUSHPULL);
```

5. Alter the GPIOs configured for AFE031 control within afe031_config.c.
   a. Alter the GPIOs configured within the HAL_afe031_cfgGpio function:

```
    //SD PIN
    GPIO_SetupPinMux(9, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(9, GPIO_OUTPUT, GPIO_PUSHPULL);
    // SD=0
    GPIO_WritePin(9, 0);
    //DAC Pin Enable
    GPIO_SetupPinMux(8, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(8, GPIO_OUTPUT, GPIO_PUSHPULL);
    // DAC=0
    GPIO_WritePin(8, 0);
```

   b. Remove or comment-out the GPIO configuration for the INT Pin within the HAL_afe031_cfgGpio function.

```
    // INT Pin - Not available on LAUNCHXL-F280049C
    // GPIO_SetupPinMux(123, GPIO_MUX_CPU1, 0);
    // GPIO_SetupPinOptions(123, GPIO_OUTPUT, GPIO_PUSHPULL);
```

6. Alter the GPIOs configured for the SPI communication within hal_spi.c:
   a. Alter the GPIOs configured within the InitAFESpiGpio function.

```
    EALLOW;
    // SPI_MOSI
    GPIO_SetupPinOptions(16, GPIO_INPUT, GPIO_ASYNC | GPIO_PULLUP);
    // SPI_MISO
    GPIO_SetupPinOptions(17, GPIO_INPUT, GPIO_ASYNC | GPIO_PULLUP);
    // SPI_CS
    GPIO_SetupPinOptions(57, GPIO_INPUT, GPIO_ASYNC | GPIO_PULLUP);
    // SPI_CLK
    GPIO_SetupPinOptions(56, GPIO_INPUT, GPIO_ASYNC | GPIO_PULLUP);
    GPIO_SetupPinMux(16, GPIO_MUX_CPU1, 1);
    GPIO_SetupPinMux(17, GPIO_MUX_CPU1, 1);
    GPIO_SetupPinMux(57, GPIO_MUX_CPU1, 1);
    GPIO_SetupPinMux(56, GPIO_MUX_CPU1, 1);
    EDIS;
```

7. Alter the GPIOs being written to for DAC enable within hal_afe031.h.
   a. Alter the GPIO written to within the HAL_afe031_dacEnable macro:

```
GPIO_WritePin(8, 1);}
```

   b. Alter the GPIO written to within the HAL_afe031_dacDisable macro:

```
GPIO_WritePin(8, 0); \
```

8. Remove the InitEPwm1Gpio() function within boostxl_afe031_f28379d_main.c

4.4.1 PWM Mode Specific Porting

The below steps are specifically for when porting boostxl_afe031_f28379d_pwmmode.

1. Change the library search path for the HRPWM library.
    a. Right click the project --> select Properties --> expand Build --> expand C2000 Linker --> select File Search Path
    b. Replace the search path for the HRPWM library from the f2837x directory to the f28004x directory.

```
${F28004X_DEVICE_SUPPORT_ROOT}/../../libraries/calibration/hrpwm/f28004x/lib
```

2. Change the volatile structure *ePWM[PWM_CH] to *ePWM[9].
3. Use ePWM6 instead of ePWM1.
    a. Replace all instances of EPwm1Regs with EPwm6Regs. (Hint: Use the Replace-all functionality within CCS.)
    b. Renaming EPWM1 functions and #defines to EPWM6 is not a requirement, but helps with code readability.

4.4.2 DAC Mode Specific Porting

The following steps are specifically for when porting boostxl_afe031_f28379d_dacmode.

1. Remove the following line of code:

```
CpuSysRegs.SECMSEL.bit.PF2SEL = 1;
```

2. Allocate RAM within the linker command file for the SINETABLE.
    a. See the default 2837x_afe031 linker command files for how to accomplish this for both RAM and FLASH configurations.
    b. Recommended to combine and allocate at least two RAM blocks for the default SINETABLE.
3. Make the following change to properly configure the F28004x SPI clock frequency.
    a. Alter the HAL_SPI_LSPCLK definition within hal_spi.h:

```
#define HAL_SPI_LSPCLK    0  //LSPCLK=SYSCLK, make 0 for f28004x
```

4. Map the pingBuf and pongBuf buffers to Global Shared RAM, if not done so already.
    a. Add the lines of code below to the top of boostxl_afe031_f28379d_main.c:

```
#pragma DATA_SECTION(pingBuf, "ramgs0");     // map pingBuf to memory
#pragma DATA_SECTION(pongBuf, "ramgs1");     // map pongBuf to memory
```

The instructions for porting the example programs to other devices and LaunchPads would be similar to those mentioned previously.

5 Receive Path

The C2000 AFE031 interface can also be used as a FSK receiver to translate the transmitted frequency shifted signal into digital data. This section describes one way to create a FSK receiver using the C2000 AFE031 interface.

5.1 Receive Path Overview

The C2000 AFE031 system receive path is shown in Figure 5-1. There is a significant amount of filtering the input signal must traverse along the path from the transformer on the right to the input of the C2000 ADC on the left. On the AFE side, the AFE031 has vast filtering abilities for these signals.

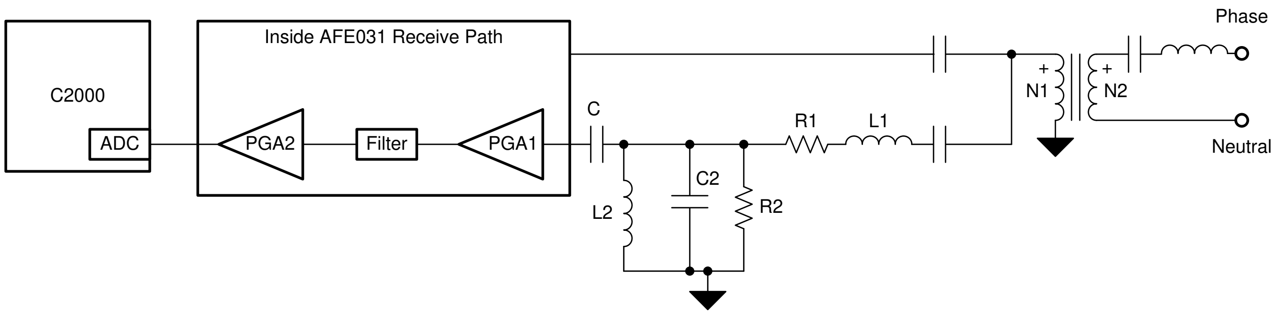


Figure 5-1. AFE031 Receive Path Interfaced With C2000 ADC

The AFE031 RX path consists of Rx PGA1, the Rx Low Pass Filter, and Rx PGA2. Both Rx PGA1 and Rx PGA2 are high performance programmable gain amplifiers that can be configured through SPI. Rx PGA1 can operate as either an attenuator, providing loss, or an amplifier, providing gain. The gain steps of the Rx PGA1 are 0.25 V/V, 0.5 V/V, 1 V/V, and 2 V/V. The gain steps of the Rx PGA2 are 1 V/V, 4 V/V, 16 V/V, and 64 V/V. For specific RX PGA Gain Select Register values, see Table 5-1. Configuring the Rx PGA1 as an attenuator (at gains less than 1 V/V) is useful for applications where the presence of large interference signals are present within the signal band. Attenuating the large interference allows these signals to pass through the analog Rx signal chain without causing an overload; the interference signal can then be processed and removed within the microcontroller, as necessary.

Table 5-1. AFE031 RX PGA Gain Settings

Bit Name	Location (0 = LSB)	Default	R/W	Function
RX1G-0, RX1G-1	0,1	0,1	R/W	This bit is used to set the gain of the RX PGA1. 00 = 0.25 V/V 01 = 0.5 V/V 10 = 1 V/V 11 = 2 V/V
RX2G-0, RX2G-1	2,3	0,0	R/W	This bit is used to set the gain of the RX PGA2. 00 = 1 V/V 01 = 4 V/V 10 = 16 V/V 11 = 64 V/V

The Rx filter is a very low noise, unity-gain, fourth-order low-pass filter. The Rx filter cutoff frequency is selectable between CENELEC A or CENELEC B,C,D modes that is set within the control register. Because the Rx filter is a very low noise analog filter, two external capacitors, shown in Figure 5-2, are required to properly configure the Rx filter. Table 5-2 shows the proper capacitance values for CENELEC A and B,C,D bands.

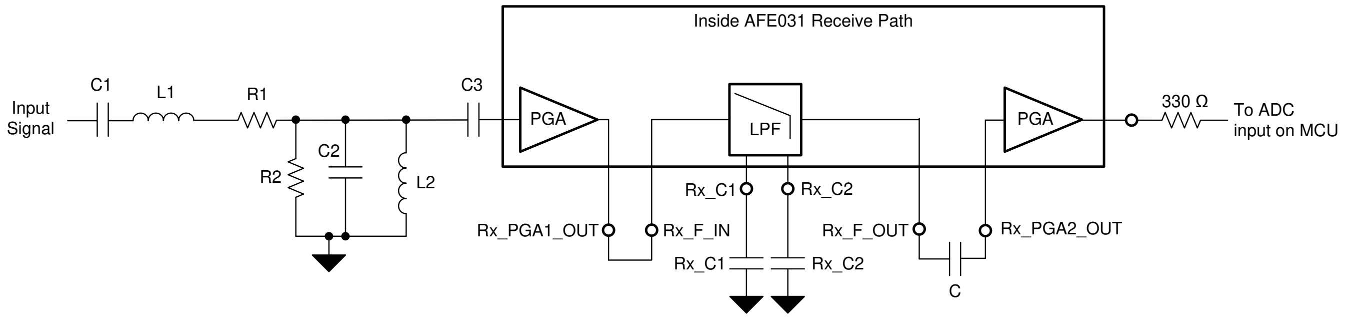


Figure 5-2. External Filtering for the AFE031 Receive Path

Table 5-2. Recommended Rx Filter External Capacitor Values

Frequency Band	Rx C1, PIN 24	Rx C2, PIN 23	Cutoff Frequency (kHz)
CENELEC A	680 pF	680 pF	90
CENELEC B,C,D	270 pF	560 pF	145

Capacitor Rx C1 is connected between pin 24 and ground, and Rx C2 is connected between pin 23 and ground. For the capacitors shown, it is recommended that these components be rated to withstand the full AVDD power-supply voltage.

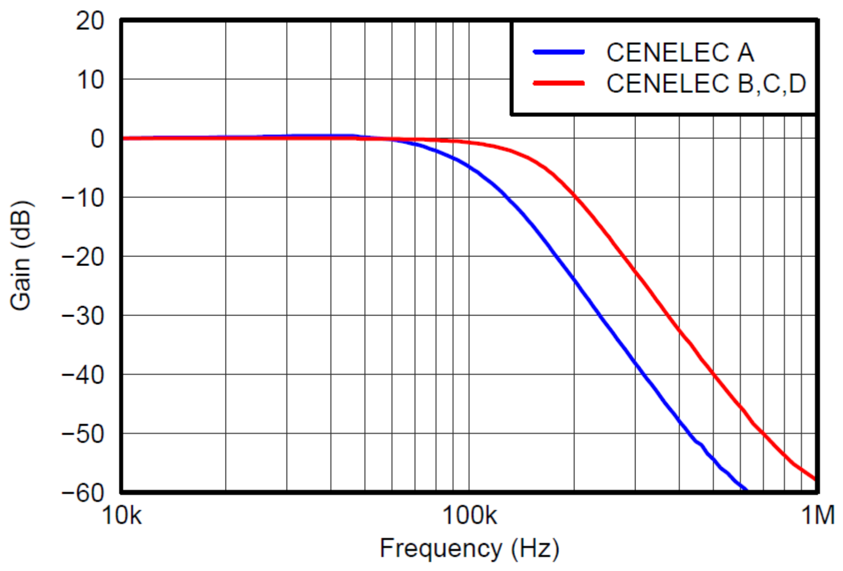


Figure 5-3. RX Filter Gain vs. Frequency

The gains witnessed at certain frequencies at the output of the RX low-pass filter are shown in Figure 5-3. This response displayed is under normal AFE03x operating conditions. Attenuation will begin at frequencies slightly before the actual cutoff frequencies in Table 5-2.

An external fourth-order passive passband filter is optional, but recommended for applications where high performance is required. The external passive passband filter removes any unwanted, out-of-band signals from the signal path, and prevents them from reaching the active internal filters within the AFE031. Table 5-3 shows the values needed for CENELEC A or CENELEC B,C,D with a 0dB passband. The component values used on the BoosterPack were for a CENELEC B,C,D with a 0dB passband.

Table 5-3. Recommended Component Values for Fourth-Order Passive Bandpass Filter (0-dB Passband Attenuation)

Frequency Band	Frequency Range (kHz)	Characteristic Impedance (Ω)	R1 (Ω)	R2 (Ω)	C1 (nF)	C2 (nF)	L1 (µH)	L2 (µH)
CENELEC A	35 to 95	1k	1k	10k	4.7	1.5	1500	4700
CENELEC B,C,D	95 to 150	1k	1k	10k	1.7	1	1200	1500
SFSK	63 to 74	1k	1	10k	2.7	2.2	2200	2200

For other bandpass filter component values with a different passband attenuation, see the *AFE031 Powerline Communications Analog Front-End Data Sheet* .

5.2 Receiver Software Implementation

Example program referenced: boostxl_afe031_f28379d_rx

The software example is designed to do the following:

- Continuously sample the received FSK signal after it has traveled through the AFE031 receive path
- Execute a correlation based algorithm on the sampled values to detect if a mark or space frequency is being received
- Decipher mark and space bits based on the frequency detected and its duration
- Store the received bits and packetize them into a desired and usable format

To enable the system to receive in software, the following needs to be completed

- The RX bit in the enable register must be set to 1 to open the AFE031 receive path.
  – Software example function: HAL_afe031_rxEnable();
- Include the fsk_corr_detect library and related header files in the project:
  – fsk_corr_detect.lib
  – fsk_corr_detector.h
  – fsk_packetization.h

5.2.1 Initial Setup and Parameters

The FSK signals being received follow a set of communication parameters that must be designed around within software.

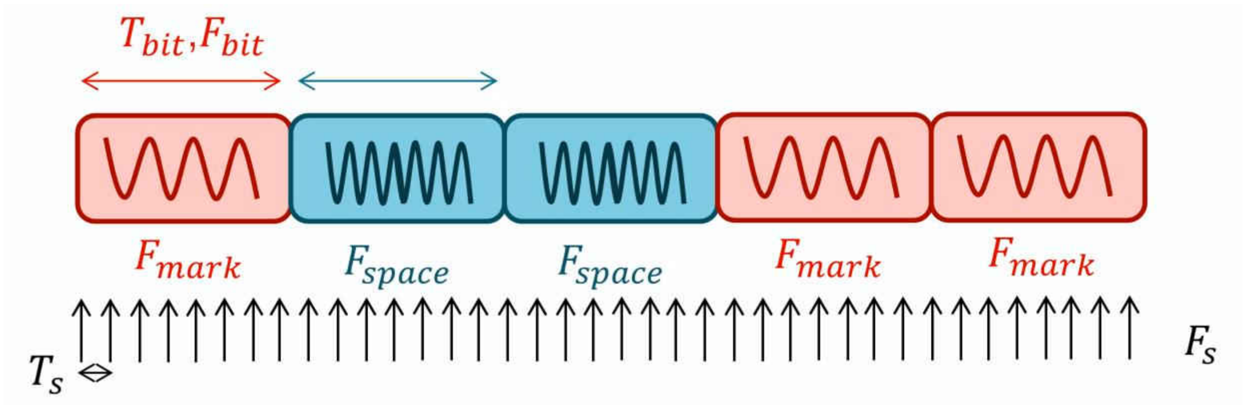


Figure 5-4. Received FSK Signal

The communication parameters of interest are illustrated in Figure 5-4. The labeled parameters represent the following:

- T_{bit} : Bit Period
- F_{bit} : Bit Frequency
- F_{mark} : Mark Frequency
- F_{space} : Space Frequency
- T_S : Sampling Period

- F_S : Sampling Frequency

For the example program the communication parameters being followed by default are stated in Table 4-1. Create a *FSK_CORR_DETECTOR* structure, declared in fsk_corr_detector.h, to hold the parameters necessary for accurate receiving.

```
volatile FSK_CORR_DETECTOR FSK_struct1; // FSK structure
```

The example software and fsk_corr_detect library are designed to detect a set of user specified frequencies, one mark frequency and one space frequency. These frequencies will have to be within the frequency band ranges of the AFE031's CENELEC A or CENELEC B,C,D configurations. The example program utilizes a mark frequency of 131.25 kHz and a space frequency of 143.75 kHz and is meant to be used with the CENELEC B,C,D configuration. Set the *mark_freq* and *space_freq* members of the *FSK_CORR_DETECTOR* structure with these frequencies.

```
FSK_struct1.mark_freq = 131250;     // Mark Frequency Detected
FSK_struct1.space_freq = 143750;    // Space Frequency Detected
```

The C2000's ADC is used to sample the FSK input signal. The sampling frequency, F_S, must follow the Nyquist theorem; the input signal must be sampled at a rate of at least 2x the highest signal frequency trying to be detected. That is, if the highest signal frequency to be detected is 100 kHz, F_S must be at least 200 kHz. In the example program the highest frequency being detected is a 143.75 kHz space frequency and the sampling rate is set to 300 kHz, which is more than the required rate. Set the *isr_freq* member of the *FSK_CORR_DETECTOR* structure to the acceptable F_S.

```
FSK_struct1.isr_freq = 300000;      // ADC Sampling frequency
```

A bit decision algorithm is intended to be run at three times the bit frequency. For example, if each bit period is 1 ms long, the bit frequency is 1 kHz making the desired bit decision frequency 3 kHz. The example program is detecting bits with a period of 5.12 ms making the bit frequency 195.3125 Hz and the desired bit decision frequency 585.9375 Hz. The bit decision frequency in software should be as close as possible to the desired frequency to prevent bit boundary issues. Set the *bit_freq* member of the *FSK_CORR_DETECTOR* structure with this bit decision frequency.

```
FSK_struct1.bit_freq = 586; // Bit decision frequency, 3x bit frequency
```

In summary, the frequency parameters set for the example program are shown in Table 5-4.

Table 5-4. Software Frequency Parameters

Parameter	Frequency
Detected Mark Frequency	131.25 kHz
Detected Space Frequency	143.75 kHz
Input Signal Sampling Frequency	300 kHz
Bit Decision Algorithm Frequency	586 Hz (rounded up)

Set the *detection_threshold* member of the *FSK_CORR_DETECTOR* structure. This value plays a role in tuning the bit detection senstivity.

```
#define FSK_BIT_DETECTION_THRESHOLD 0.1 // Bit detection threshold value
FSK_struct1.detection_threshold = FSK_BIT_DETECTION_THRESHOLD; // Set threshold
```

Complete the fsk_corr_detect library's initialization based on the member values inputted by calling the corresponding init function.

```
FSK_CORR_DETECTOR_INIT(&FSK_struct1); // Initialize FSK structure
```

Additionally, the format of received information is taken into account by setting the following parameters within software.

- The number of bits that make up a word, #define within fsk_packetization.h:
  - #define NUMBER_OF_BITS_PER_WORD 11
- The number of words that make up a packet, #define within fsk_packetization.h:
  - #define NUMBER_OF_WORDS 3
- The number of total bits within a packet, #define within fsk_corr_detector.h:
  - #define RX_MESSAGE_SIZE 33

5.2.2 Interrupt Service Routines

ISRs running at these predetermined frequencies carry out the main functions of the receiver solution. The example program makes use of the C2000's EPwms and CPU timer to trigger necessary interrupts.

An ADC sampling ISR, configured using EPwm1, is set to trigger at the 300 kHz input signal sampling frequency. The ISR function samples the ADC, scales the sampled value, and passes the scaled value to a library function for signal processing.

A bit-decision ISR, configured using EPwm2, is set to trigger at 585.92 Hz. This frequency is as close to the desired 585.9375 Hz the EPwm module could achieve. The ISR function takes the signal processing work of the previous ADC sampling ISR and checks if a mark or space bit has been detected. If a mark or space bit is detected then the detected bit is placed into a received message buffer. Once the message buffer is full, a flag will be set to signify a full packet has been received.

A message timeout ISR, configured using CPU timer 2, is set to trigger if the user specified time limit is reached while receiving. By default this time limit is set to 3 seconds by the RX_MESSAGE_TIMEOUT #define. The timer begins right before the system starts receiving and will reset if a packet is received. If a packet is not received within the specified time limit, the ISR triggers and causes the system to stop receiving.

5.2.3 Run Time Operation

During run time the receiver operates in the following way:

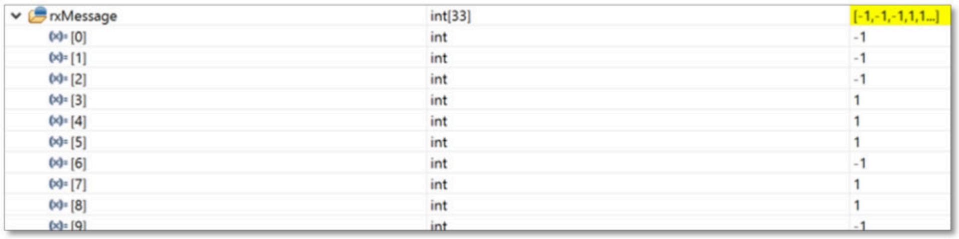
1. The system will begins to continuously receive incoming data by starting the EPwms and CPU timer to trigger interrupts.
   a. Example Program Function: *Start_Receiving();*
2. The ISRs will run until the full packet of data is received or the specified timeout is reached.
   a. The *rxMessage*[] buffer will be filled with the received bits while the ISRs are running
3. Once the packet of data is received or the specified timeout is reached, the system will stop receiving data to reduce CPU utilization during quiet periods.
   a. Example Program Function: *Stop_Receiving();*
   b. The function stops the EPwms and stops and resets the CPU timer
4. The received data is then packetized into the desired format.
   a. Example Program Function: *Packetize(int message[], int packet[]);*
   b. The function takes the *rxMessage[]* buffer containing 33 received bits and fills the *packet[]* buffer with three, 11 bit, words by summing up the received bits for each word. The *rxMessage[]* buffer contents are then set to zero.
      i. +1 equates to a W1 following the specs in Table 4-1
      ii. -1 equates to a W0 following the specs in Table 4-1
   c. The function sums up the values of the *packet[]* and saves the sum to the *packet_sum* variable
      i. +3 equates to a packet_1 following the specs in Table 4-1
      ii. -3 equates to a packet_0 following the specs in Table 4-1

5. The packetized data is used before the receive process restarts.
    a. Example Program Function: *Visual_Indication();*
    b. The function causes the LEDs on the BOOSTXL-AFE031 to blink accordingly
        i. Blue LED blinks if a packet_1 was received
        ii. Red LED blinks if a packet_0 was received
        iii. Red and Blue LEDs blink if neither a packet_1 or packet_0 were received, or if a timeout occurs

5.2.4 Testing Results

The performance of the FSK receiver solution was tested in a controlled lab environment using either the PWM or DAC mode transmitter solution to provide the input FSK signal. The established setup is similar to what is shown in Figure 2-8 without any coupling circuitry. The purpose of the tests were to confirm that the receiver solution could effectively translate a received FSK signal into the original digital information.

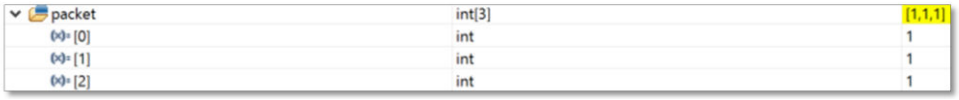
When the guidelines discussed in Section 5.2 are appropriately followed, a C2000 MCU can take a FSK input signal and accurately decode each bit of data it contains. This is shown in Figure 5-5, where a packet of thirty-three unique bits were successfully captured.



rxMessage	int[33]	[-1,-1,-1,1,1...]
[0]	int	-1
[1]	int	-1
[2]	int	-1
[3]	int	1
[4]	int	1
[5]	int	1
[6]	int	-1
[7]	int	1
[8]	int	1
[9]	int	-1

Figure 5-5. Packet of Recieved Bits of Data

This captured data is in the form of an 11-bit code word transmitted three consecutive times. Each code word having a sum of one is then packetized in the format shown in Figure 5-6



packet	int[3]	[1,1,1]
[0]	int	1
[1]	int	1
[2]	int	1

Figure 5-6. Packetized Data

5.2.5 System Utilization

The resources consumed by the C2000 MCU when being used as an FSK receiver are listed in Table 5-5.

Table 5-5. C2000 Resources Utilized

Resource Name	Type	Purpose	Usage/Restrictions
ADCINA	Module/IO	ADC input for sampling input signal	Restricted to ADC inputs that are available an accessible
EPwm1	Module	Triggers the interrupt for the signal sampling routine	Can be configured a number of ways to meet desired ISR frequency
EPwm2	Module	Triggers the interrupt for running the bit decision routine	Can be configured a number of ways to meet desired ISR frequency
SPI	Module/IO	For accessing the AFE03x's registers during initialization	Only needed for initialization for the RX solution
CPU Timer 2	Module	Can be used to create a timeout when no information has been received for a certain amount of time	ISR may have lower priority than other set ISRs

The number of CPU cycles consumed by each fsk_corr_detect library function are listed in Table 5-6.

Table 5-6. Library Function CPU Cycles

Function Name	Description	CPU Cycles	Type
FSK_CORR_DETECTOR_INIT	Initializes variables used by the FSK library, based off the frequency parameters the user sets within a fsk_corr_detector structure	60	Initialization
FSK_CORR_DETECTOR_ RUN	Performs necessary calculations on the sampled ADC values to demodulate the input signal	59	Run Time
FSK_CORR_DETECTOR_ OverSampl_RUN	Performs the logic to decipher if a bit has been received	134	Run Time
Packetize	Takes a received message data buffer and builds usable code words and packets	1381	Run Time

The run-time CPU utilization of the software solution, when being used in its default state and receiving information specified in Table 4-1, can be calculated using the information in Table 5-7.

Table 5-7. Software ISR/Function Usage

ISR/Function	Average Cycles	Frequency of Execution
ADC Sampling ISR	74	300 kHz
Bit-decision ISR	175	Approximately 586 Hz
Packetization Function	1381	Approximately 1 Hz

CPU Utilization Equation:

CPU Utilization = $((74 \cdot F_S + 175 \cdot 3 \cdot F_{bit} + 1381 \cdot F_{Packetization}) / F_{CPU}) \cdot 100\%$

CPU Utilization at F28379D's 200 MHz Clocking Frequency:

CPU Utilization =((74*300kHz + 175*586Hz + 1381*1Hz) / 200MHz)*100% = 11.15%

5.2.6 Device Dependency and Porting

While the FSK receiver solution was built for and tested on the F2837xD, it should directly port to devices that have Floating Point Unit (FPU) and Trigonometric Math Unit (TMU) support. The fsk_corr_detect library's functionality is dependent on the FPU and TMU being present. Devices with both FPU and TMU support include: F2837xD, F2837xS, F2807x, and F28004x.

5.3 Tuning and Calibration

The C2000 AFE031 system can be tuned and calibrated for receiving within software and hardware. These concepts are discussed in the following subsections.

5.3.1 Setting the AFE03X's PGAs

The AFE031's PGAs along the RX path are used to amplify or attenuate the input signal to be within the desired voltage range before entering the C2000's ADC. It is desired for the signal at the ADC pin to be within the range defined by the ADC's reference voltage. For example, the F28379D LaunchPad's reference voltage is 3V; therefore, the signal should be within a range of 0 V to 3 V to prevent clipping.

For best results, this signal should be close to this voltage range, without going over or under, to utilize the full resolution of the ADC. Use the AFE031's PGAs to tune the signal to meet this criteria using the gain value settings in Figure 5-1. For how to set the RX PGA values within software, see step 10 of Section 3.1.

5.3.2 Automatic Gain Control (AGC)

If the receiver system is to be used within an application that has an input signal with inconsistent amplitudes, it may be necessary to implement some form of Automatic Gain Control (AGC) to manage the signal before it reaches the C2000's ADC input. The AGC would be a closed-loop feedback system that measures the amplitude of the input signal, or a related response, and based on what is observed amplify or attenuate the signal.

The AFE031's internal PGAs could be leveraged to dynamically amplify or attenuate the signal to be within the desired amplitude range. For gain select times of the two PGAs along the RX path, see the AFE031 Datasheet .

Another approach would be to perform the AGC externally before the signal enters the C2000/AFE031 system. However, this requires additional circuitry involving a variable gain amplifier (VGA) to be added.

5.3.3 Setting the Bit Detection Threshold

The *detection_threshold* member of the FSK_CORR_DETECTOR structure, mentioned in Section 5.2.1, is in direct relation to the accuracy of bit detections. The threshold needs to be set to a value between 0 and 1. If set to a value too high then bits will not be detected at all, but if set too low then there could be confusion as to what constitutes a mark or space bit and cause bit errors. It may take some trial and error to find a value that consistently detects bits without errors.

5.3.4 FSK Correlation Detector Library

For inquiries related to the FSK correlation detector library utilized in the software example, email the following address: C2000-fsk_rx_source_access@list.ti.com.

To request access to the library's source code (contingent on end application review and export control approval from Texas Instruments Incorporated), follow the instructions at the following url: https://www.ti.com/licreg/docs/swlicexportcontrol.tsp?form_id=268791&prod_no=C2000-AFE031_FSK_RX_SOURCE&ref_url=c2000.

5.4 Porting RX to LAUNCHXL-F280049C

Below are the steps that need to be taken to port the boostxl_afe031_f28379d_rx from the LAUNCHXL-F28379D to the LAUNCHXL-F280049C after importing the project into Code Composer Studio.

1. Perform steps 1 through 7 in Section 4.4.
2. Make the following changes to the ConfigureADC(void) function within boost_afe031_f28379d_rx_main.c.
   a. Remove the line of code below:

```
AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); // 12 bit res
```

   b. Add the line of code below before the EALLOW;

```
//
// Setup VREF as internal
//
SetVREF(ADC_ADCA, ADC_INTERNAL, ADC_VREF3P3);
```

3. Setup the ADC for ePWM triggered conversions on channel 9 instead of 1 within boost_afe031_f28379d_rx_main.c.
   a. Replace the SetupADCEpwm() function call with the line of code below:

```
SetupADCEpwm(9);
```

4. Configure the CPU timer for a 100 MHz clock instead of 200 MHz.
   a. Replace the ConfigCpuTimer() function call with the line of code below:

```
ConfigCpuTimer(&CpuTimer2, 100, RX_MESSAGE_TIMEOUT);
```

5. Allocate Global Shared RAM within the linker command file for fsk_corr_lib_data.
   a. See the default 2837x_afe031 linker command files for how to accomplish this for both RAM and FLASH configurations.

The instructions for porting the example program to other devices and LaunchPads would be similar to those above.

6 Interfacing With a Power Line

Power Line Communication (PLC) is utilized with Texas Instruments C2000 paired with the AFE031 device, which enables data to be sent over existing power cables. This means that one can both power and control/ retrieve data at the same time with just power cables running though these devices. This minimizes the overall cost that would be needed otherwise to create a communications path with extra cabling.

6.1 Line Coupling

Line coupling is one of the most crucial segments of the PLC system having two primary functions. The first function is to couple the signal from the AFE031 to and from the ac mains/dc Buss. The second is to prevent the low frequency high voltage of 50/60 Hz from the mains from damaging the PLC circuitry.

6.2 Coupling to an AC Line

For coupling to the ac main, the following components are needed: low-voltage capacitor, transformer, high-voltage capacitor, and an inductor. It is important to note that this section does not show the necessary protection circuitry; that information is discussed in Section 6.4.

A simplified diagram is shown Figure 6-1.

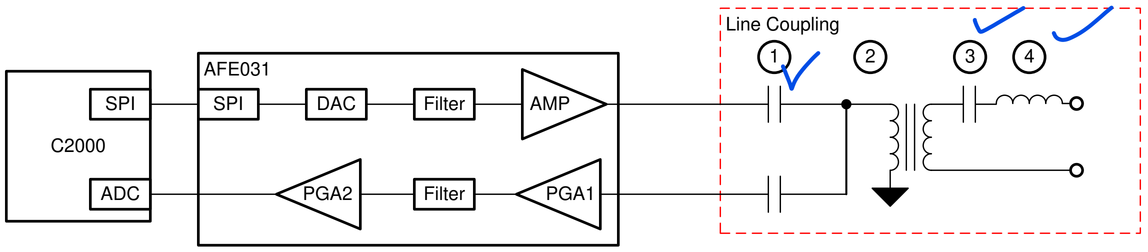


Figure 6-1. Simplified Line Coupling Circuit for AC Main

This section addresses the components from the left side (that connects to the TX and RX of the AFE031 device) to the right side (that connects to the power line). Be sure to consider the following key points during hardware design.

6.2.1 Low Voltage Capacitor

The low-voltage (LV) capacitor couples the time-varying components of the PA output signal into the line coupling transformer. The LV capacitor should have a large enough capacitance to appear as a low impedance throughout the signal band of interest. A value of 10 µF is a common value for signals ranging from 35 kHz to 150 kHz. The boostpack LV capacitor value was 10 µF because the space and mark frequencies (131.25 kHz and 141.75 kHz) lie within the range. The voltage rating of the LV capacitor should be sufficient to withstand the clamping voltage of the TVS diode (discussed in the protection section) operating under surge conditions. Generally, this limit should be equal or slightly higher to the PA supply voltage.

6.2.2 The Ratio of the Transformer

Most power-line communication transformers are compact, with turns ratios between 1:1 and 4:1, low leakage inductance, and approximately 1-mH of winding inductance. This inductance in series with the high-voltage capacitor results in a voltage divider attenuating the ac mains voltage down to negligible levels at the module output.

To determine the optimal turns ratio for the transformer it must be based on the PA's capabilities of maximum output swing and maximum output current to achieve maximum power transfer into the load.

$$\frac{N_1}{N_2} = \sqrt{\frac{PA, V_{OUT\_Peak}}{PA, I_{OUT\_Peak} \times R_{Load}}} \qquad (2)$$

There are three cases where a one limitation is dominant than the others.

- **Case 1:**

  If the turns ratio of the transformer is greater than the ideal calculated value, the TX output of the AFE031 is limited by the voltage swing of the PA.
- **Case 2:**

  If the turns ratio of the transformer is less than the ideal calculated value, the TX output of the AFE031 is limited by the maximum output current from the PA.
- **Case 3:**

  If the turns ratio of the transformer is equal to the ideal calculated value, the TX maximum output occurs as the amplifier approaches both its maximum output voltage and maximum output current resulting in maximum power transfer to the load.

It is also important to be aware that the transformer affects the coupling emission performance of the EN50065-1 when under a 2-MHz frequency. In order to compensate, TI recommends using products from Wurth Electronik.

6.2.3 HV Capacitor

The high-voltage (HV) capacitor blocks the low-frequency mains voltage by forming a voltage divider with the winding inductance of the line coupling transformer. While using high voltage CBB capacitors, the maximum voltage range must exceed the amplitude of the power grid ac voltage. Operating the capacitor at approximately 80% of its ac-rated voltage ensures a long component operating life. The next important requirement is a standard setting maximum reactive power (VA limit). For example, the European product standards for attaching a device to the power grid must have a reactive power of less than 10 VAR, resulting in a capacitor values less than .55 µF. The equation below shows how the value .55 µF was used to determine the HV capacitor value.

$$HVCap = \frac{VA_{LIMIT}}{VAC^2\left(2\pi \times f\right)} \tag{3}$$

For a 240-VAC, 50-Hz application with a 10-VA limit,

$$HVCap \le \frac{10}{240^2\left(2\pi \times 50\right)} \cong 550\ nF \tag{4}$$

It is important to note that a metallized polypropylene electromagnetic interference and radio frequency interference (EMI/RFI) suppression capacitor is recommended because of the low loss factor associated with the dielectric, which results in minimal internal self-heating.

6.2.4 HV Side Inductor

The inductor that is connected in series with the HV capacitor is required when driving low line impedances and the HV Cap is restricted to approximately 470 nF, for the reasons previously stated. In applications that operate in the Cenelec A band, the impedance of the 470-nF capacitance at 40 kHz is approximately 8.5 Ω. If the application requires the ability to drive a 2-Ω load, for example, this series impedance is restrictive. Adding the series inductor can mitigate this effect. To properly select the value of the inductance, the operating frequency range of the system must be known. A common example would be the PRIME frequency band, which is approximately 40 kHz to 90 kHz. Selecting the HV Cap and inductor to have a resonant frequency in the center of the frequency band is recommended, and results in a series inductor value of 12.8 µH and HV Cap value of 470 nF.

$$L = \frac{1}{HVCap \times \left(2\pi \times f\right)^2} \tag{5}$$

The inductor should be sized to be capable of withstanding the maximum load current without saturation.

6.3 Coupling to DC Line

Coupling to a dc line has similar components for protection with the lack of components such as HV inductor, MOV, and the transformer. Figure 6-2 shows what is needed for coupling circuit and transient protection.

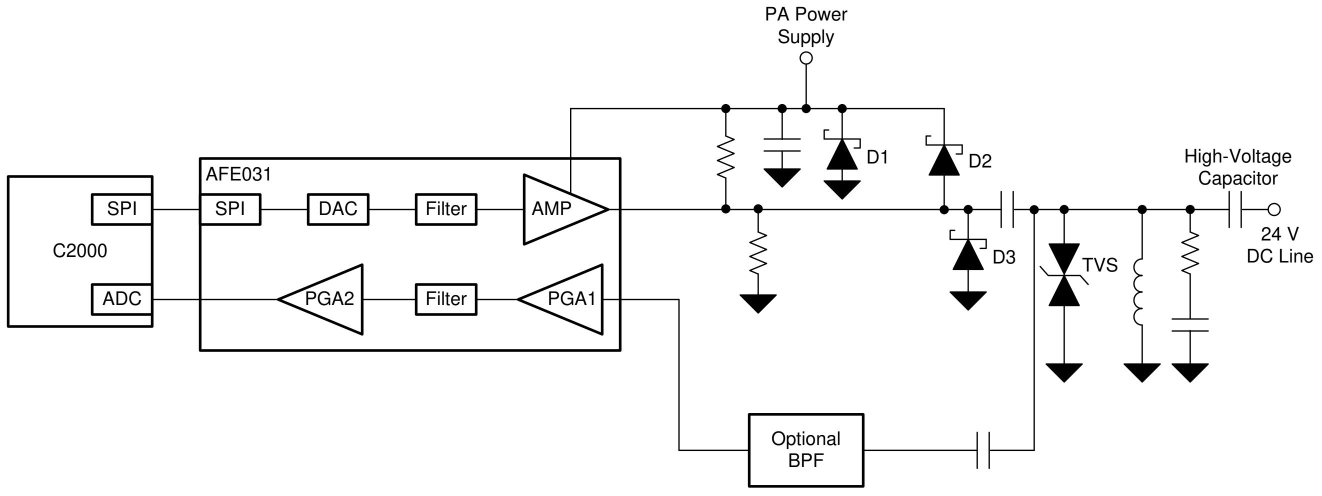


Figure 6-2. Coupling Circuit for a DC Line

A 10 µF capacitor was used to couple to the dc line. When coupling to a dc line it is important to remember that line is generally low impedance and may affect the output swing of the PA in the AFE031. Connecting an inductor in series with the line wikk provided enough impedance to the power line communications signal such that the power supply (possibly very low impedance) does not interfere with the PLC signal modulation. Figure 6-3 shows PLC modules coupled to a low-impedance dc line.

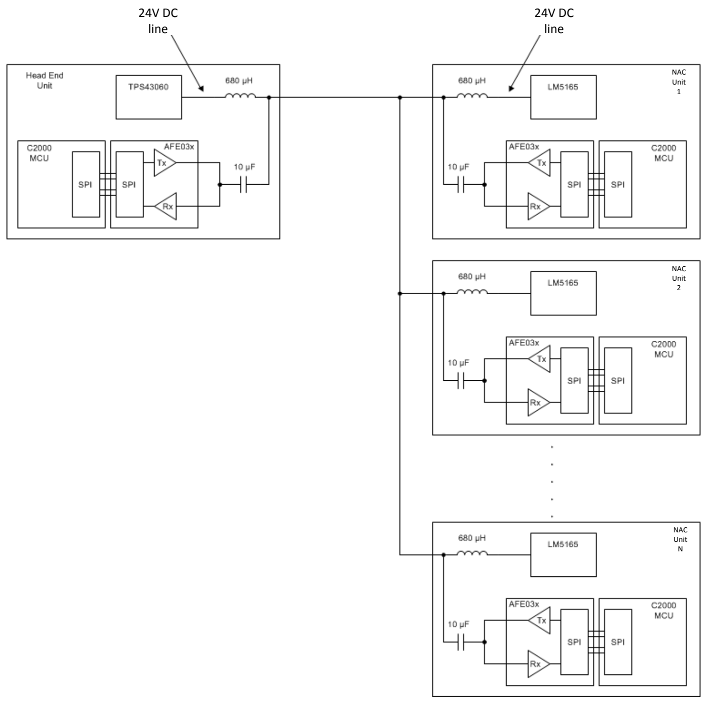


Figure 6-3. Example of Multiple PLC Modules Coupled to a DC Line

For detailed test data on this circuit, see the *DC Power-Line Communication Reference Design*.

6.4 Protection Circuit

Powerline communications are often located in operating environments that are harsh for electrical components connected to the ac line. Noise or surges from electrical anomalies such as lightning, capacitor bank switching, inductive switching, or other grid fault conditions can damage high-performance integrated circuits if they are not properly protected. The AFE031 can survive even the harshest conditions if several recommendations are followed: metal-oxide varistors (MOVs), transient voltage suppression diodes (TVSs), Schottky diodes, and a Zener diode.

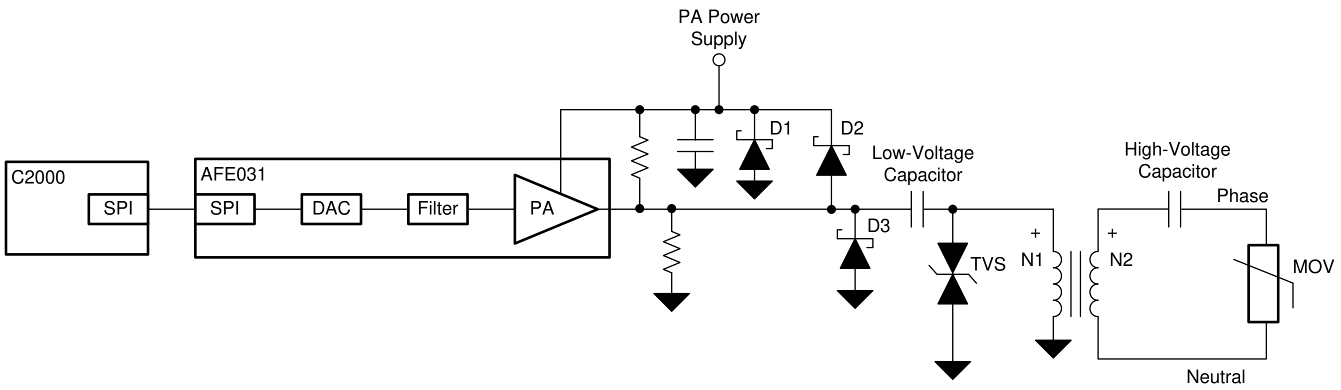


Figure 6-4. Recommended Transient Protection

6.4.1 Metal Oxide Varistors

There are several factors to consider when selecting an MOV: Working voltage, required amount of transient energy to be absorbed by the MOV, peak transient current, and power dissipation.

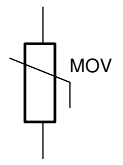


Figure 6-5. Metal Oxide Varistor (MOV)

A MOV is a device that has high resistance until its triggering voltage is exceeded. Once this voltage level has been exceeded, the MOV reduces its resistance and absorbs the energy from the pulse. The I/V characteristic of a typical MOV is shown in Figure 6-6.

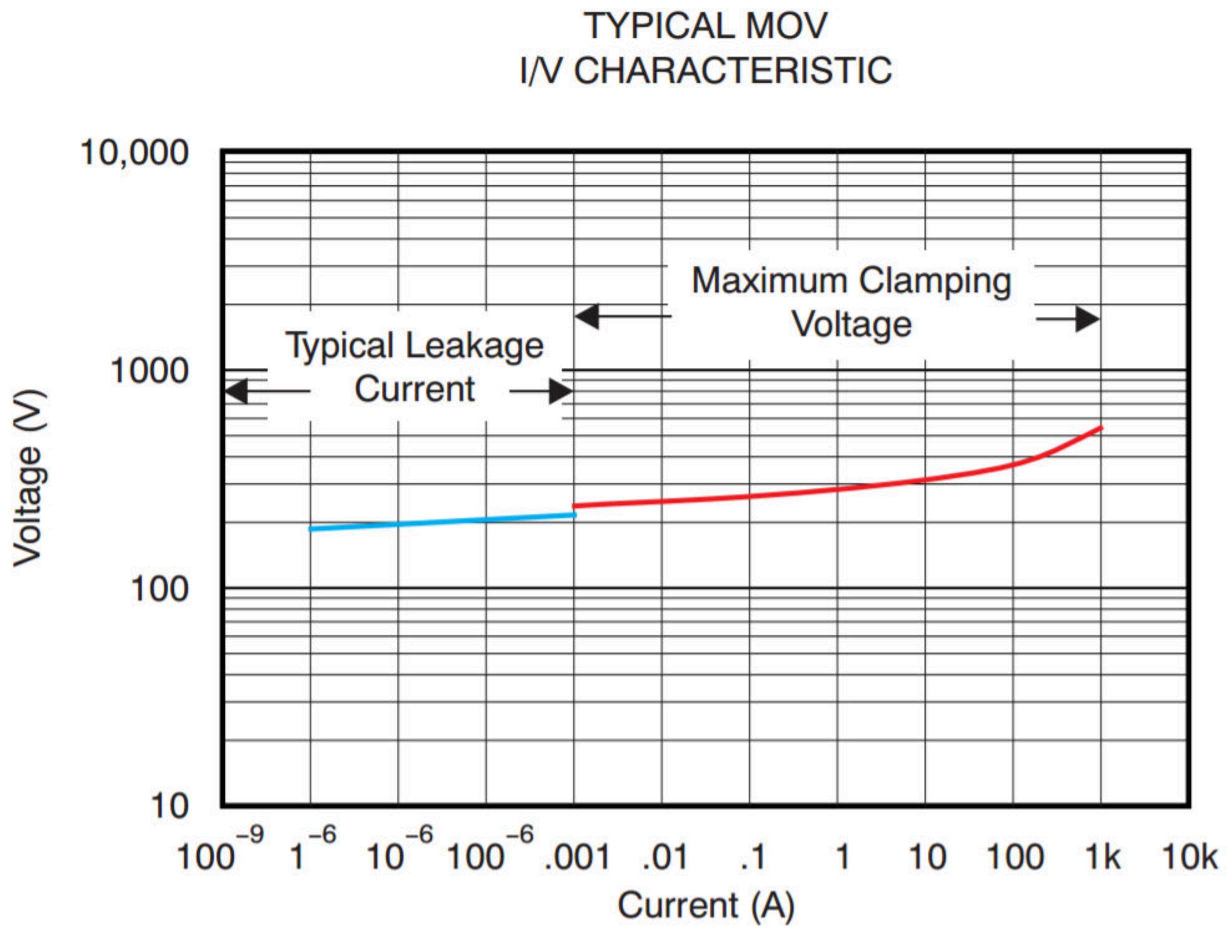


Figure 6-6. Typical MOV I/V Characteristic

By the nature of the materials and techniques used in the construction of these components, MOVs respond quickly to a fast transient pulse, have high instantaneous power ratings, and are well-suited for protection on the ac line. The maximum clamping voltages are typically specified in response to a high-speed transient similar to that shown in Figure 6-7. The 8/20 µs waveform is commonly associated as a waveform that represents the spectral content of lightning strikes.

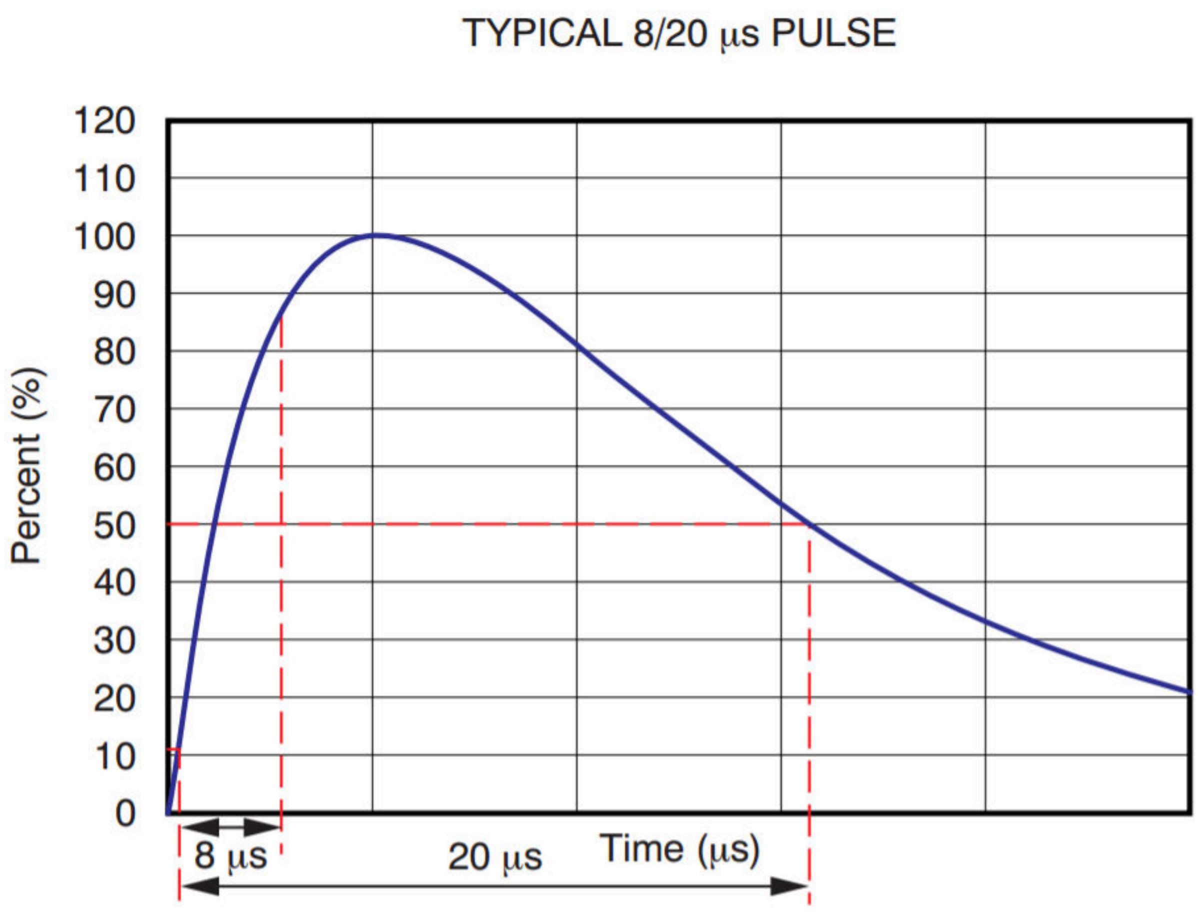


Figure 6-7. Typical 8/20-µs Pulse for MOV and TVS Performance Specification

6.4.2 Transient Voltage Suppressors

A TVS is a very fast-acting clamping device that turns on in the case of an overvoltage condition, shunting the surge of current into ground. TVSs are rated primarily by the power handling capability and the clamping voltage. TVSs are available in either unidirectional or bi-directional configurations.

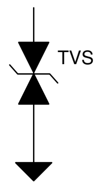


Figure 6-8. Bi-Directional TVS Diode

For power-line communication applications, a bi-directional TVS is recommended; the component is placed next to the line coupling transformer. Figure 6-9 shows the I/V characteristics for a typical bi-directional TVS.

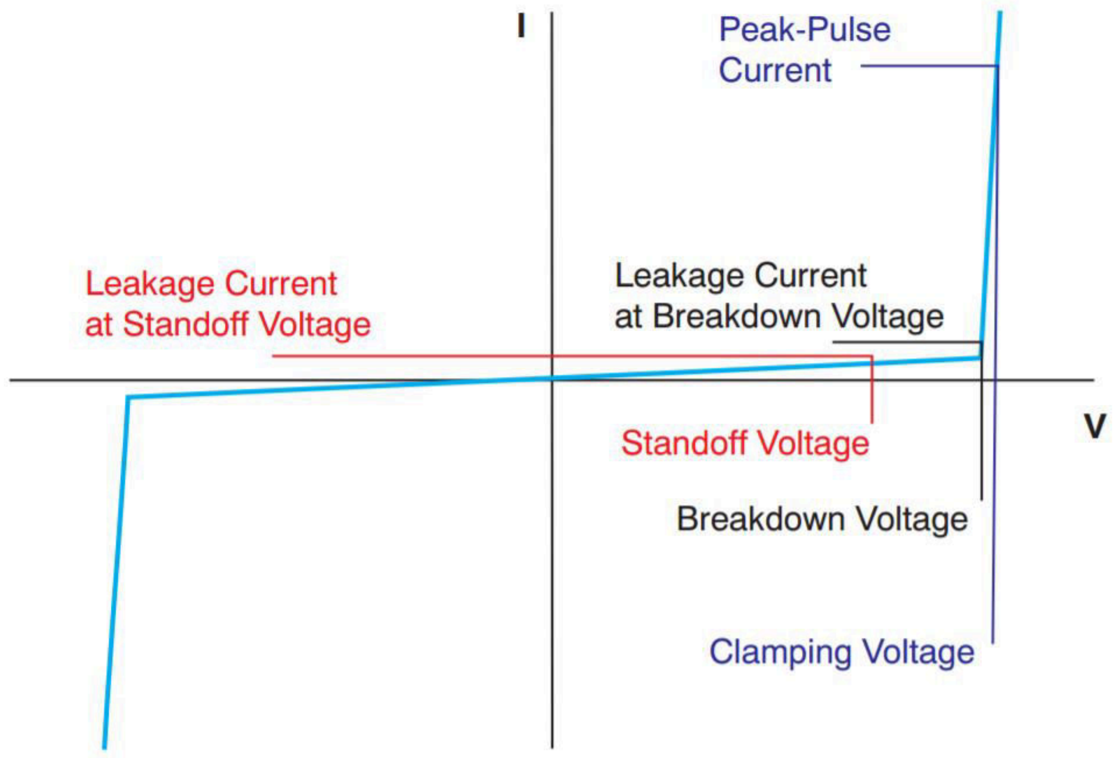


Figure 6-9. Typical Bi-Directional TVS I/V Characteristics

As a surge or pulse on the ac line occurs, the voltage rises across the TVS. If the voltage rises higher than the TVS breakdown voltage, the TVS turns on and rapidly changes from high impedance to low impedance, shunting current into ground. Note that the low-voltage capacitor between the PA output and the TVS blocks any dc voltage at the TVS. As a result, the normal FSK or OFDM signal from the PA appears to be centered around ground at the TVS. This condition requires the TVS to be bi-directional.

Figure 6-10 illustrates this concept.

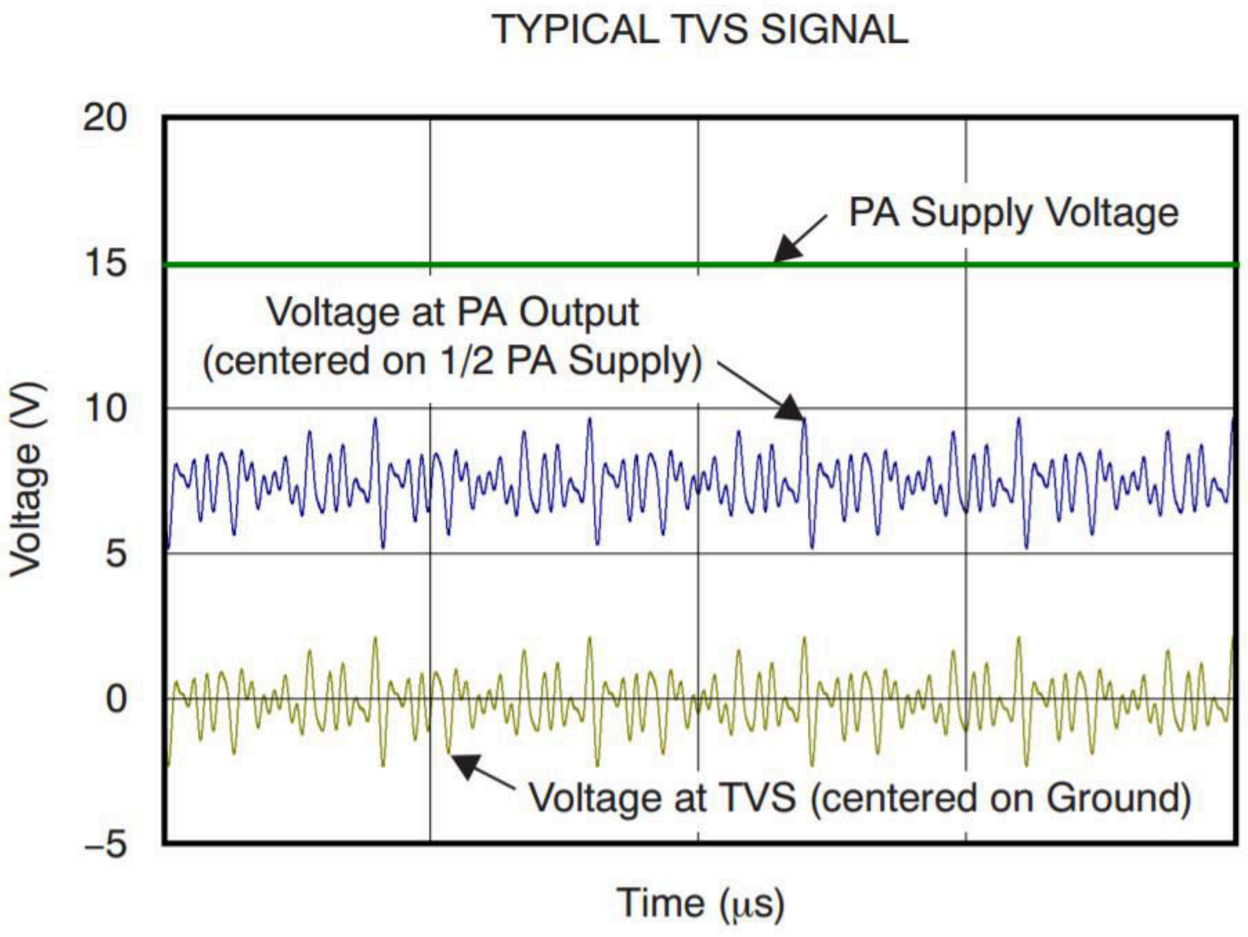


Figure 6-10. Typical Signal at the TVS

Because the signal is symmetric around ground at the TVS, the TVS breakdown voltage should be equal to approximately one-half of the PA power-supply voltage. It is important for the TVS to remain off during normal operation to avoid clipping and introducing distortions to the output signal. It is also important that the TVS turn on and clamp at the lowest possible voltage beyond normal operation to provide maximum protection.

This BoosterPack was designed with surge protection parts. A bi-directional transient-voltage suppression (TVS) diode with a shunt connection to attach on the LV side of the transformer is used to clamp the voltage of a surge. The stable voltage of the TVS must be exactly 1/2 of the PA supply voltage of the AFE, meaning a 15-V AFE must use a 7.5-V TVS. This 1/2 ratio is based on the fact that:

- The PA of the AFE is a type AB, which uses a single power rail. So, the 12-V powered PA TX output is at a 6-V bias with a ±6-V amplitude, meaning that the signal has a ±6-V range. While on transmission, the TVS must not be allowed to saturate the signal, so the stable voltage must be kept to ≥ ½ of the PA power rail.
- If a 12-V PA uses a 7.5-V TVS, during the arrival of a surge pulse, the LV side bias is locked-on at 7.5 V, but if the surge occurs at the exact moment of a TX maximum amplitude (which is 6 V), then the signal on the TX route is 7.5 V + 6 V = 13.5 V, which is higher than the power rail and causes damage to the PA of the AFE. So, the TVS voltage must be ≤ 1/2 of the PA power rail.

6.4.3 Current Steering Diodes

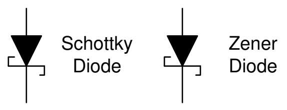


Figure 6-11. Current Steering Diodes

While the MOV and TVS components clamp the pulse and either dissipate or re-direct most of the energy to ground, it is also recommended to place current-steering (Schottky) diodes at the output of the PA section of the AFE031. In the unlikely event a transient surge increases the PA output pin beyond its power-supply rail, low-drop Schottky diodes can steer the current around the AFE031 safely to ground. Maintaining a low (less than 0.8 V) forward voltage drop on the Schottky diode is recommended for maximum protection. If the Schottky diode that connects the output of the PA to the power-supply rail turns on and becomes forward-biased, it is important to steer the current to ground without significantly disturbing the PA power-supply voltage. Placing a zener diode at the PA power-supply pins to ground provides a low-impedance path for surges that attempt to raise the power-supply voltage beyond the absolute maximum rated voltage for the AFE031.

6.5 Determining PA Power Supply Requirements

Calculating the minimum power-supply requirements for the PA, the desired load voltage, load impedance, and available power-supply voltage or desired transformer ratio are all the parameters that must be known. For this FSK power-line communication example, similar to PRIME, the goal is to drive a 1-VRMS signal into a 2-Ω load. The minimum power-supply voltage required was calculated by adding the peak-to-peak load voltage; the voltage dropped across the HV Cap and inductor, V2; the voltage dropped across the LV Cap, V1; and twice the output swing to rail limit of the PA, VSWING. For FSK and SFSK systems, the peak-to-average ratio is √2, while this ratio is approximately 3:1 for OFDM systems.

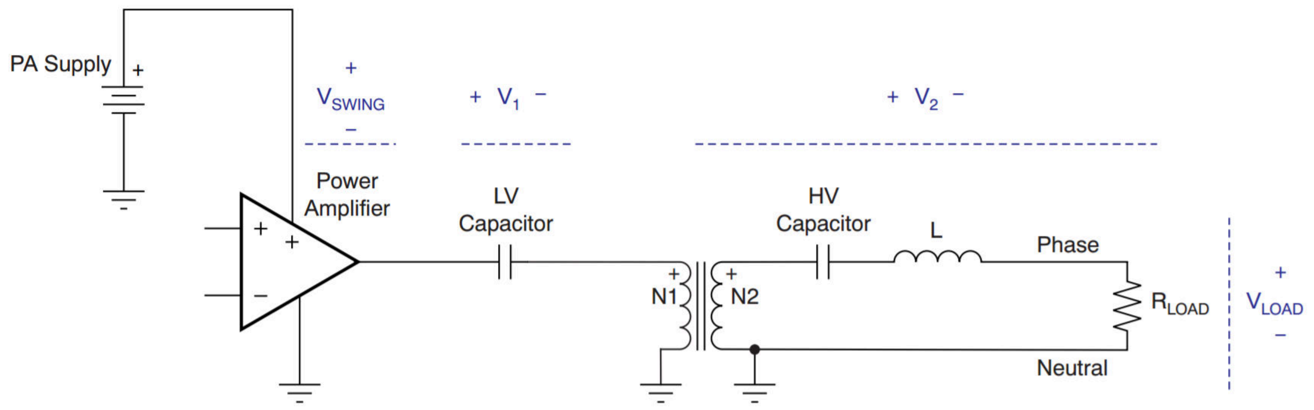


Figure 6-12. Typical Line Coupling Circuit

These ratios must be considered when performing calculations that relate the RMS voltages and peak voltages during an analysis. Choosing a large value for the LV Cap results in the voltage drop (V1) becoming negligible in most circumstances. The losses in the transformer are also negligible, even at high load currents, if the proper transformer with a low DCR is used. For FSK and SFSK systems, the voltage drop across the HV Cap and inductor, V2, is also usually negligible; in OFDM systems, because of the wider operating bandwidth, voltage drop V2 can be ignored and accounted for by using a 1.5x multiplier on the load voltage as an approximation.

For the AFE031 BoosterPack an FSK signal with a 2-Ω load and 1-VRMS load voltage:

$$PA_{Supply} = V_{Load} \times Turns\ Ratio \times \left(2 \times V_{Swing}\right) \tag{6}$$

$$PA_{Supply} = 2.878\ V \times 1.5 + \left(2 \times 2\ V\right) = 8.25\ V \tag{7}$$

For OFDM, the following equation can be used:

$$PA_{Supply} = V_{Load} \times OFDM \times Multiplier \times Turns\ Ratio \times \left(2 \times V_{Swing}\right) \tag{8}$$

For more information on Power dissipation of the AFE031, see *Analog Front-End Design for a Narrowband Power-Line Communications Modem Using the AFE031*.

7 Summary

The FSK transmitter provides a stepping stone for communications development using an AFE030/1 device with a C2000 F28379D. More development could be done to demonstrate the system as a FSK receiver. By providing this example, the hope is that someone who is interested in PLC communications can pick it up and begin learning. By creating hardware that is flexible enough to be used in multiple modes allows various communications protocols to be implemented on the same hardware. As a final reminder, all software and hardware can be found in C2000Ware.

8 References

- SunSpec Protocol:
  - SunSpec Home page
- AFE031 Information and Designs:
  - *Analog Front-End Design for a Narrowband Power-Line Communications Modem Using the AFE031*
  - *AFE031 Powerline Communications Analog Front-End Data Sheet*
  - *DC Power-Line Communcation Reference Design*
  - *LAUNCHXL-F28379D Overview*

9 Schematics

9.1 Schematics (PWM Mode)

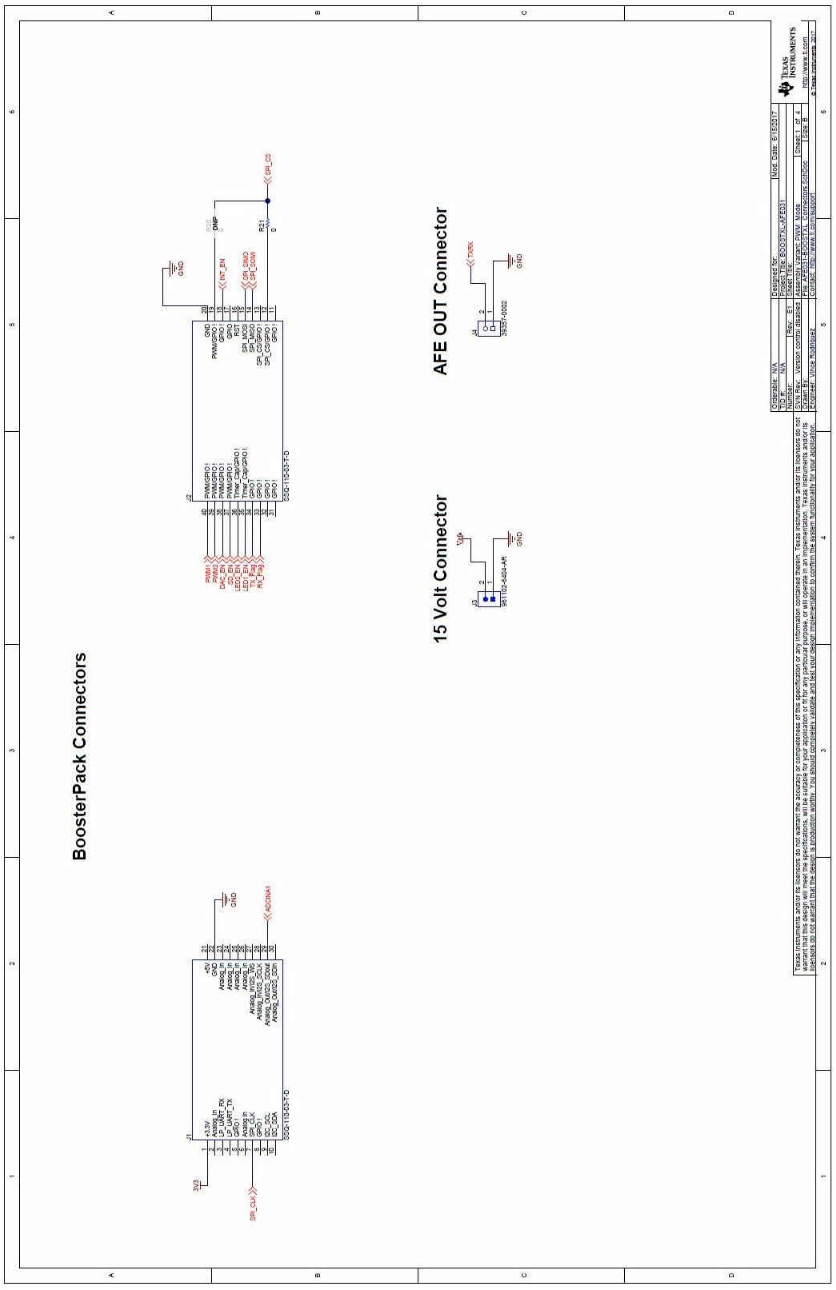


Figure 9-1. Schematic PWM Mode - Page 1

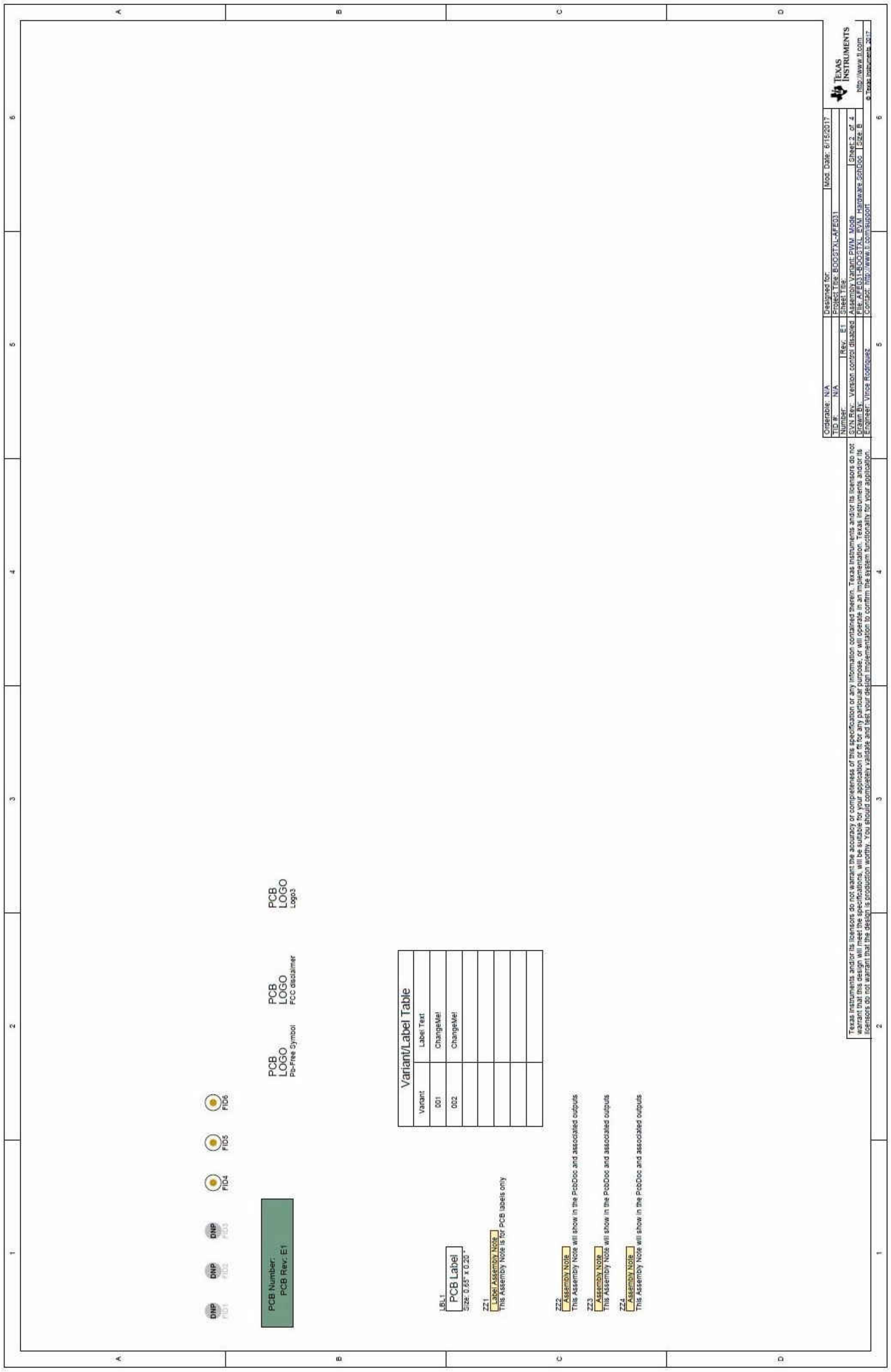


Figure 9-2. Schematic PWM Mode - Page 2

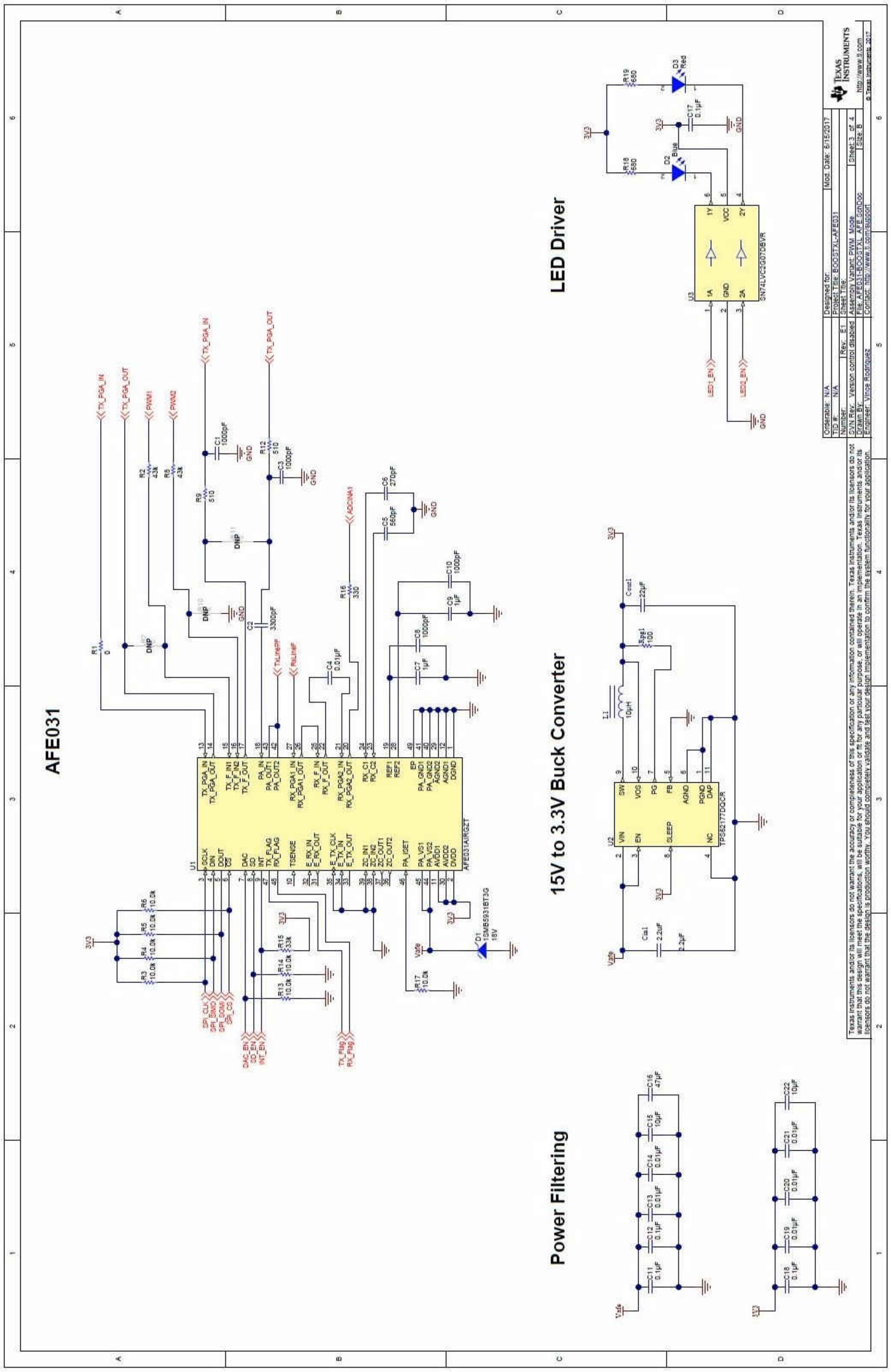


Figure 9-3. Schematic PWM Mode - Page 3

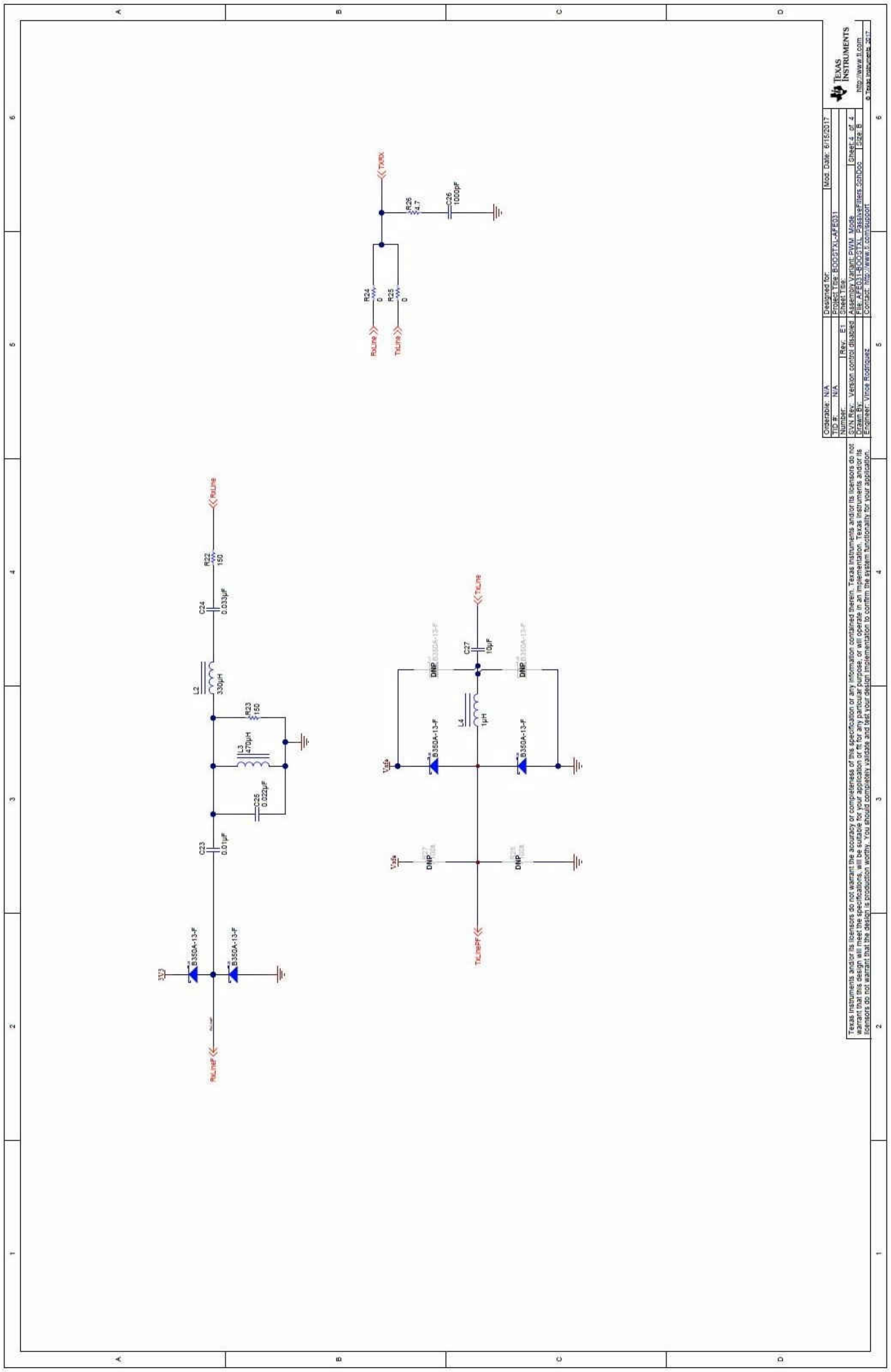


Figure 9-4. Schematic PWM Mode - Page 4

9.2 Schematics (DAC Mode)

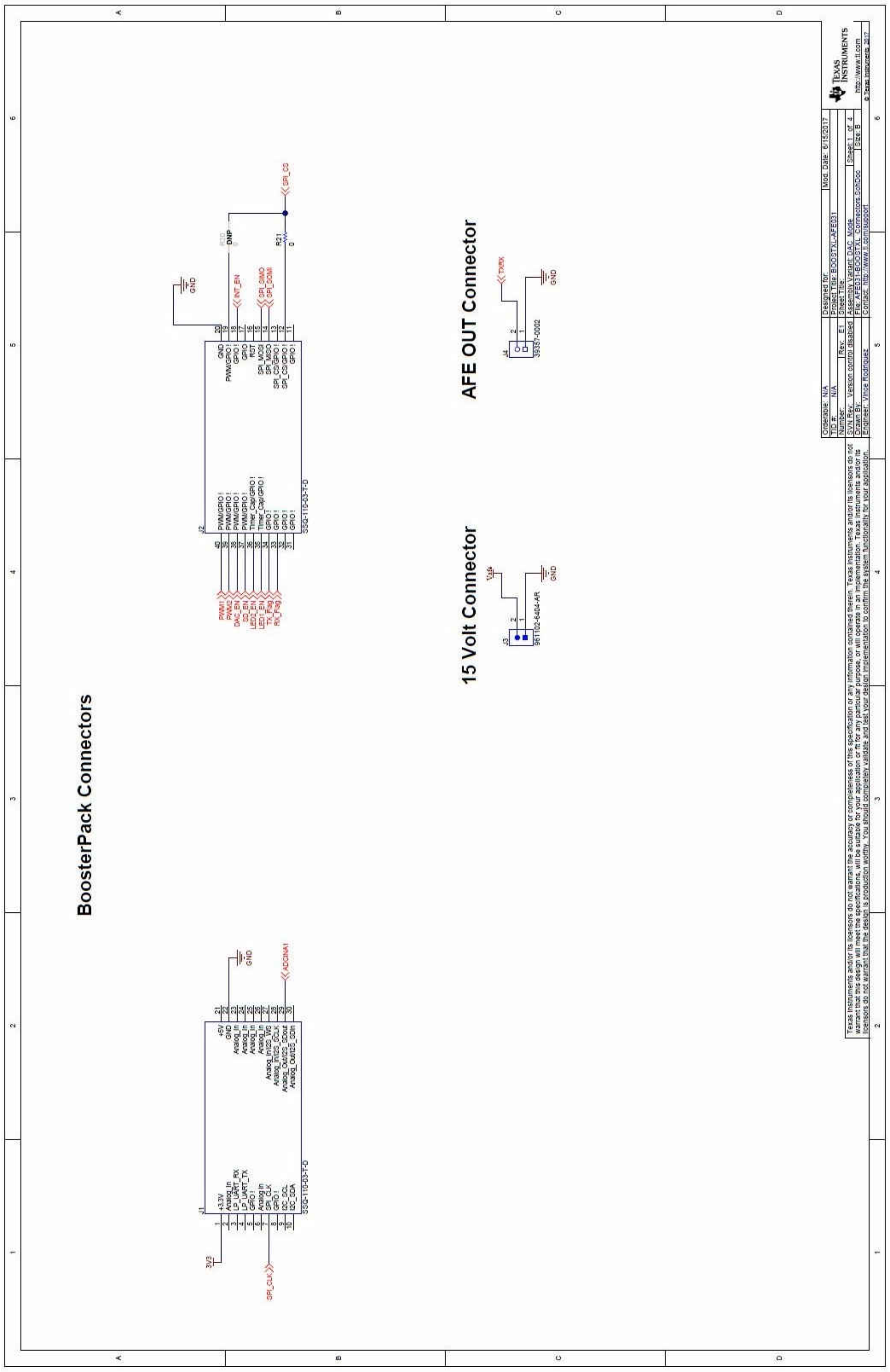


Figure 9-5. Schematic DAC Mode - Page 1

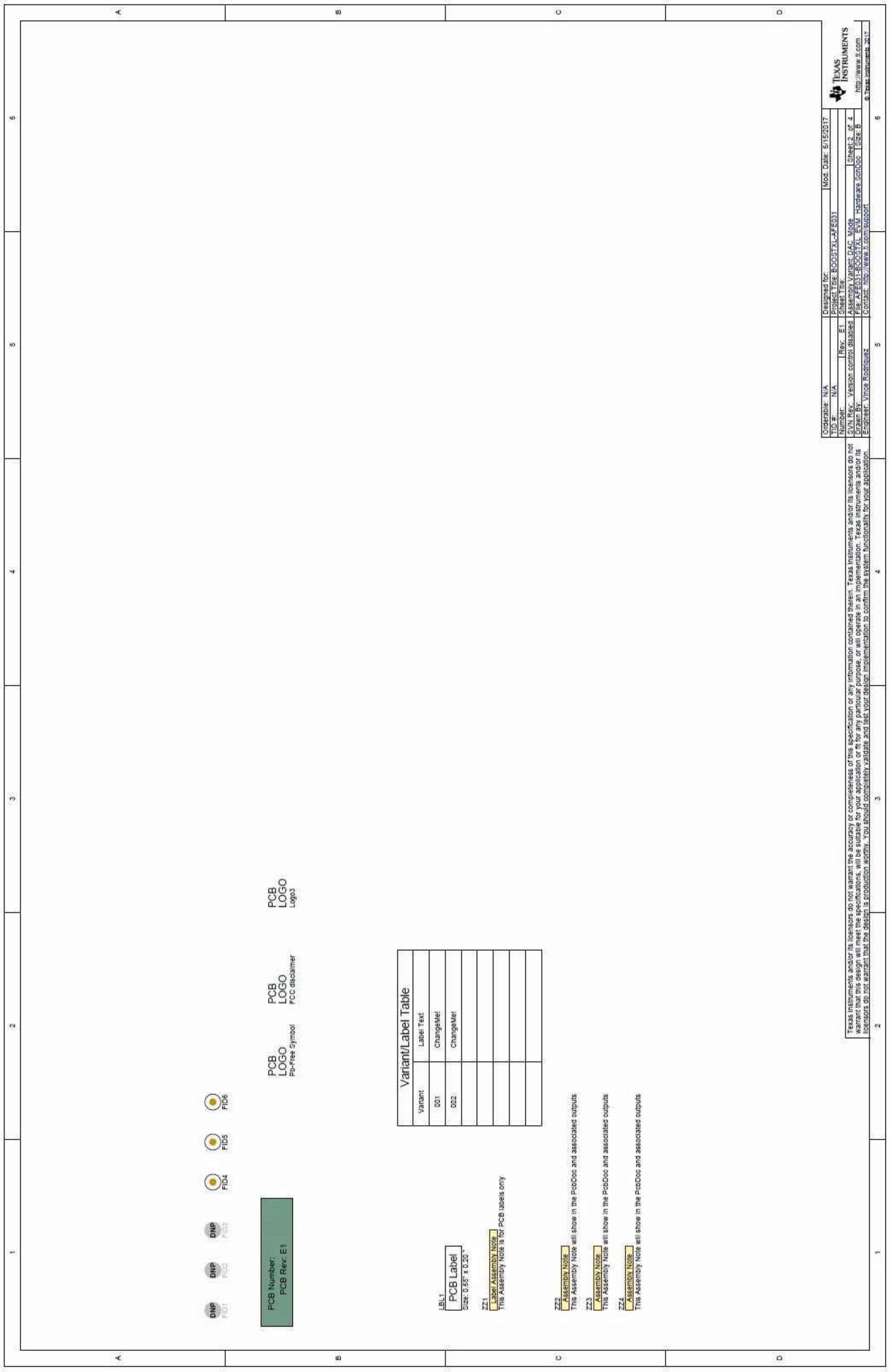

PCB Number:
PCB Rev: E1
DNP FID1
DNP FID2
DNP FID3
FID4
FID5
FID6
PCB LOGO Pb-Free Symbol
PCB LOGO FCC disclaimer
PCB LOGO Logo3
Variant/Label Table
Variant
Label Text
001
ChangeMe!
002
ChangeMe!
LBL1
PCB Label
Size: 0.65" x 0.20"
ZZ1
Label Assembly Note
This Assembly Note is for PCB labels only
ZZ2
Assembly Note
This Assembly Note will show in the PcbDoc and associated outputs
ZZ3
Assembly Note
This Assembly Note will show in the PcbDoc and associated outputs
ZZ4
Assembly Note
This Assembly Note will show in the PcbDoc and associated outputs
Orderable: N/A
TID #: N/A
Number:
Rev: E1
SVN Rev: Version control disabled
Drawn By:
Engineer: Vince Rodriguez
Designed for:
Project Title: BOOSTXL-AFE031
Sheet Title:
Assembly Variant: DAC_Mode
File: AFE031-BOOSTXL_EVM_Hardware.SchDoc
Contact: http://www.ti.com/support
Mod. Date: 6/15/2017
Sheet: 2 of 4
Size: B
Texas Instruments
http://www.ti.com
© Texas Instruments 2017
Texas Instruments and/or its licensors do not warrant the accuracy or completeness of this specification or any information contained therein. Texas Instruments and/or its licensors do not warrant that this design will meet the specifications, will be suitable for your application or fit for any particular purpose, or will operate in an implementation. Texas Instruments and/or its licensors do not warrant that the design is production worthy. You should completely validate and test your design implementation to confirm the system functionality for your application.
A
B
C
D
1
2
3
4
5
6

Figure 9-6. Schematic DAC Mode - Page 2

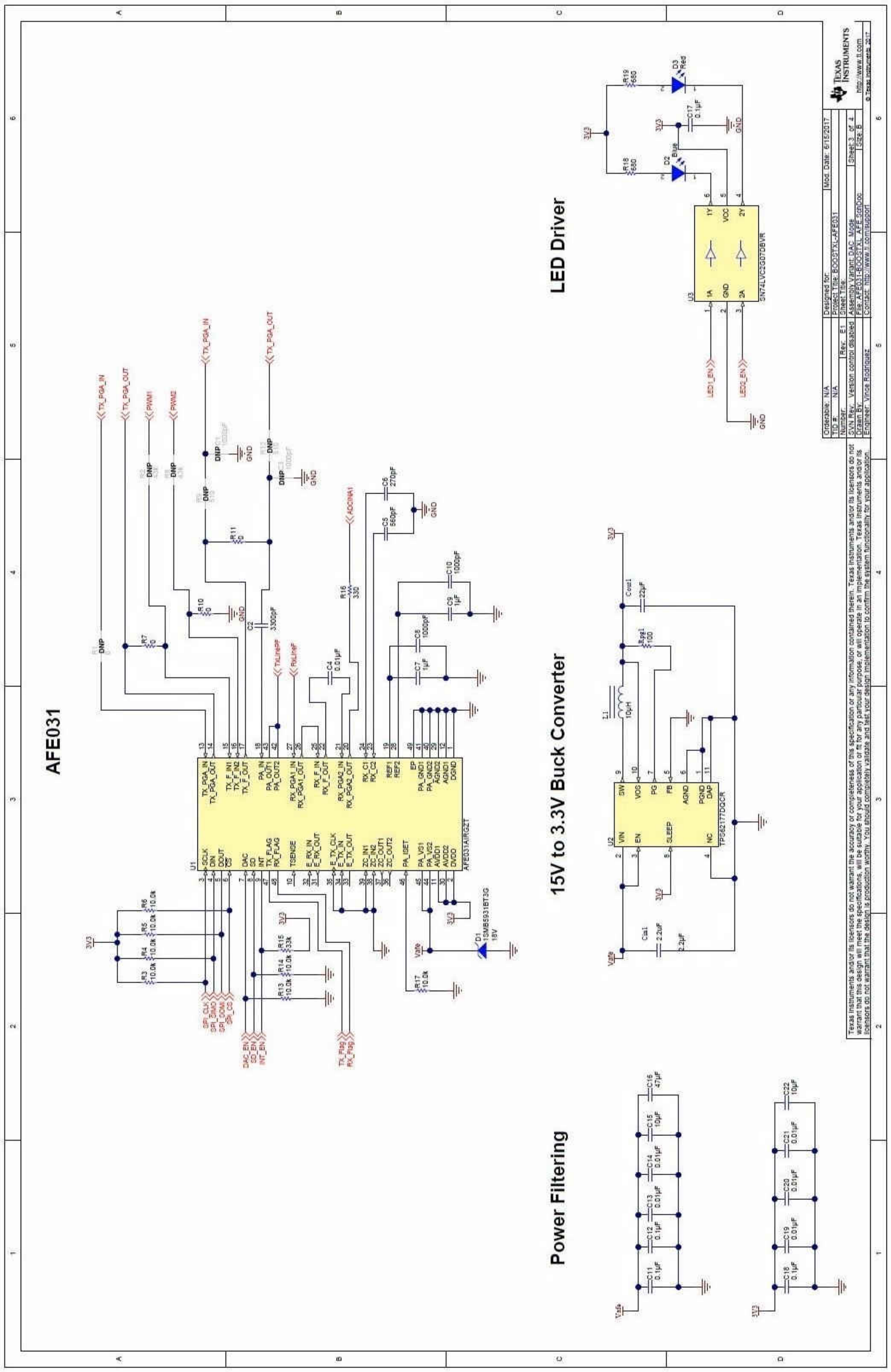


Figure 9-7. Schematic DAC Mode - Page 3

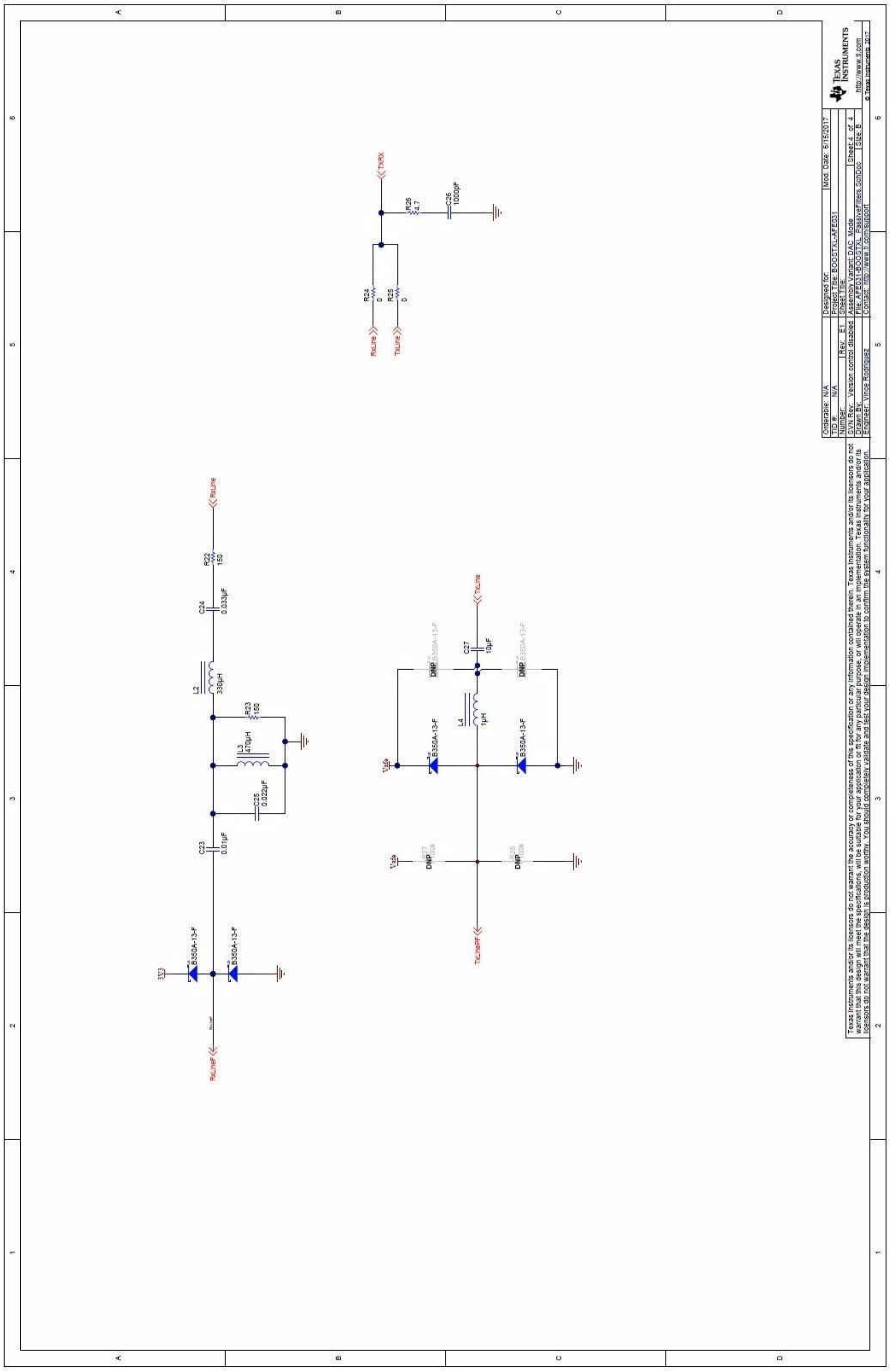


Figure 9-8. Schematic DAC Mode - Page 4

10 Revision History

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

Changes from Revision C (September 2018) to Revision D (March 2022)	**Page**
• Updated the numbering format for tables, figures, and cross-references throughout the publication................	1

Changes from Revision B (January 2018) to Revision C (September 2018)	**Page**
• Updates were made in Section 4.4..	21
• Updates were made in Section 5.4..	32