

A Low-Cost 12-Bit Speech CODEC Design Using the MSP430F13x

Kes Tam, Vincent Chan

MSP/MS

ABSTRACT

This application report illustrates a design of a single chip speech codec (ADC and DAC) using the MSP430F133 MCU from Texas Instruments. The ADC part of this codec uses the on chip 12-bit ADC. The DAC part uses a novel PWM technique which enables the DAC to operate at a high over-sampling frequency with high resolution. This simplifies the subsequent filter circuitry. Combined with on chip voice compression algorithms, this solution forms a cost effective and ultra-low-power base band circuit for voice communication applications.

Contents

1	Introduction	2
2	System Overview	3
3	Filters response	4
3.1	Moving Average and Comb Filter	4
3.2	Forth-Order LPF	5
3.3	Combined Response.....	7
4	Oversampling	9
5	Dual PWM	9
6	Summary.....	10
	References	11
	Appendix A. Schematic.....	12
	Appendix B. Software Listing	13

Figures

Figure 1.	System Block Block Diagram	3
Figure 2.	Moving Average or Comb Filter Gain Response	5
Figure 3.	A Second Order Low-Pass Filter	6
Figure 4.	Output Analog LPF Gain Response	7
Figure 5.	Ouput Section Gain Response	8
Figure 6.	Overall System Gain Response.....	8
Figure 7.	Combing Two PWM Outputs.....	10

1 Introduction

The MSP430F13x series MCU from Texas Instruments (hereafter referred to as F13x) has a built-in 12 bit high quality ADC, which is well suited for voice as well as instrumentation applications. However, for many voice communication applications, a DAC is also required.

The F13x, like many other MCUs, uses its PWM modules as a DAC. The challenge here is to maintain the power consumption level, and therefore the clock speed, to a minimum while providing good DAC resolution at a high over-sampling frequency. The high over-sampling frequency is required to simplify the post filtering circuitry and therefore to keep down cost circuit space.

Section 2 shows the system block diagram of this design from audio input, through different digital and analog sampling and filtering stages, to the final audio output.

We have used digital filters with over-sampling to simplify the hardware. Section 3 describes the digital and analog filters used in this system. Section 4 talks about the over-sampling technique used.

This design uses a PWM DAC of 12 bits resolution at 40 ksps (5 times over-sampling) running from an 8 MHz clock. These contrasting demands are met by using a novel dual PWM technique. This technique is described in detail under Section 5.

The hardware schematic and software listing is included in the appendices. It can be easily constructed using Texas Instruments' MSP-FET430P140 development tools.

2 System Overview

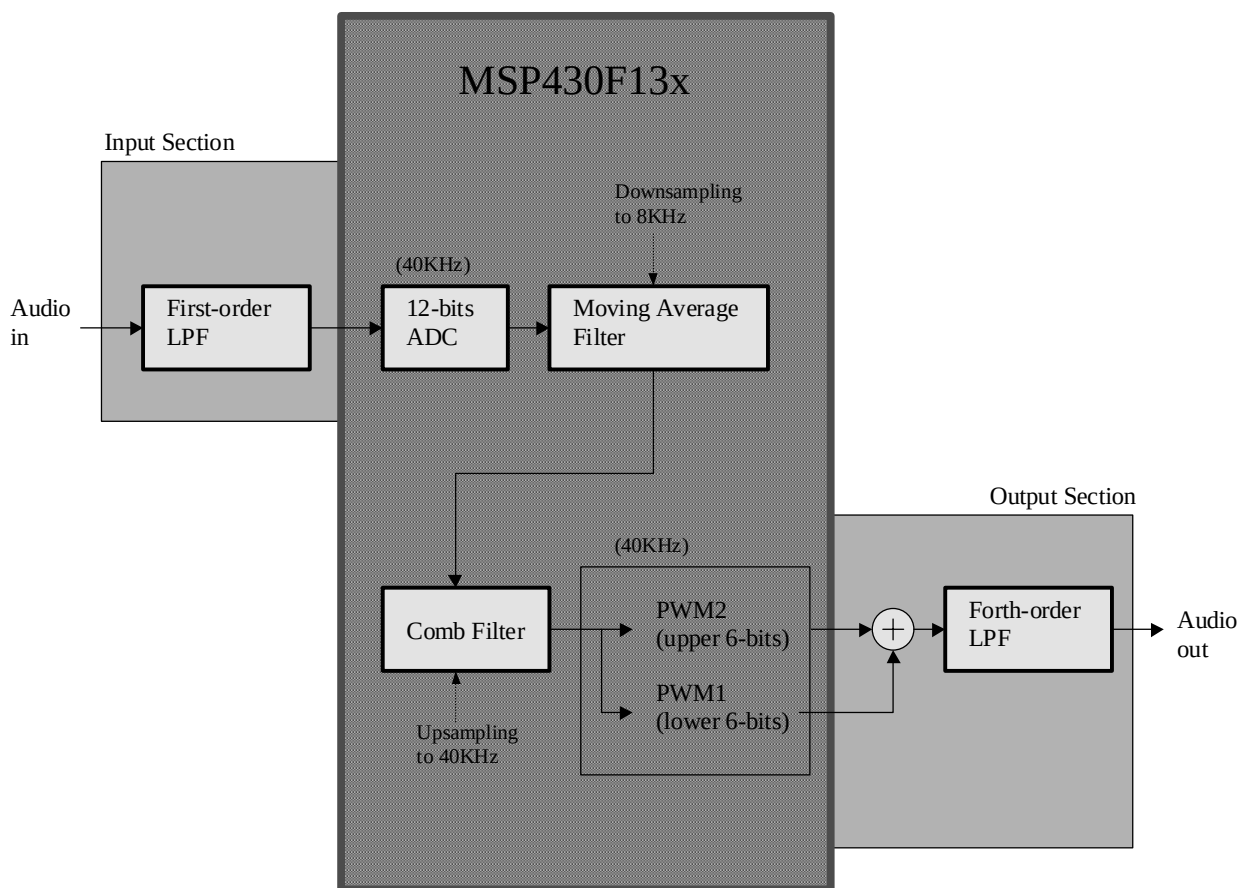


Figure 1. System Block Diagram

Figure 1 shows the system block diagram. The voice signal is first over-sampled at 40 kps and then internally down sampled to 8 kps. The frequency response of the codec is designed to have a very fast roll off at approximate 3.5 kHz.

For easy understanding, the whole system is represented as input to output signal chain.

The input section consists of three stages.

- The first stage provides first-order pre-sample low pass filtering and electret microphone amplification. The roll off is set to start at 4 kHz, and because the signal is over-sampled at 40 kps, the roll off of this low pass filter can be very gentle.
- The second stage provides the analog to digital conversion using the on chip 12-bit ADC of the F13x device. The built-in 16 word auto-scan buffer of the ADC allows multiple samples to be accessed simultaneously. This economizes on CPU ADC access bandwidth.

- c) The third stage consists of a moving average filter and a down sampler from 40 kps to 8 kps. The cut off frequency of this digital filter is set to 4 kHz.

For demonstration purpose, the output data of the input section is internally routed to the output section. In real applications, the sampled data is often further processed before being sent through a network. For example, for wireless voice communication, data is compressed using ADPCM coding before being sent out.

The output section consists also of three stages.

- a) In the first stage, samples come in at 8 kps and go out at 40 kps. In between is an up-sampler and a comb filter.
- b) The next stage is the PWM DAC process. In this stage, two 6 bit PWMs makes up a 12-bit DAC. More detail will be covered in Section 5.
- c) The last stage is a forth-order analog low pass filter which converts the digital output of the PWM to an analog signal. The 40-kHz carrier is filtered out.

3 Filters response

A total of four filters are used, two analog and two digital. The first stage low-pass filter has been covered in Section 2. We will cover the comb filter and the moving average filter in detail here. The forth-order LPF (low pass filter) is made up of two second-order LPFs cascaded together. They are made identical to simplify the design.

3.1 Moving Average and Comb Filter

Moving average filters and comb filters does not provide very fast roll off. However, they are very simple filters to implement.

The transfer function of the comb filter used here is given by

$$H(z) = (1/k) * (1 - z^{-k}) / (1 - z^{-1}) \quad (1)$$

Where:

$$f_c = f_s / k$$

$$f_c = \text{cut off frequency}$$

$$f_s = \text{sampling frequency}$$

In actual implementation, the output sequence can be obtained with the equation below.

$$y[n] = y[n-1] + (x[n] - x[n-k]) / k \quad (2)$$

where y is the output and x is the input

The moving average filter comes out the same as the comb filter. The output of the moving average of k inputs is given by:

$$y[n] = (x[n] + x[n-1] + \dots + x[n-(k-1)]) / k \quad (3)$$

For next output, it will be:

$$y[n+1] = (x[n+1] + x[n] + \dots + x[(n+1)-(k-1)]) / k$$

$$\text{or } y[n+1] = (x[n+1] + k * y[n] - x[n-(k-1)]) / k$$

$$\text{or } y[n+1] = y[n] + (x[n+1] - x[n-(k-1)]) / k$$

If we put n back with $n-1$, we get

$$y[n] = y[n-1] + (x[n] - x[n-k]) / k$$

which is the same equation as (2).

This means the moving average filter is the same as the comb filter. To have a cutoff frequency of 4 kHz, we choose $k=10$ and the response is shown in Figure 2.

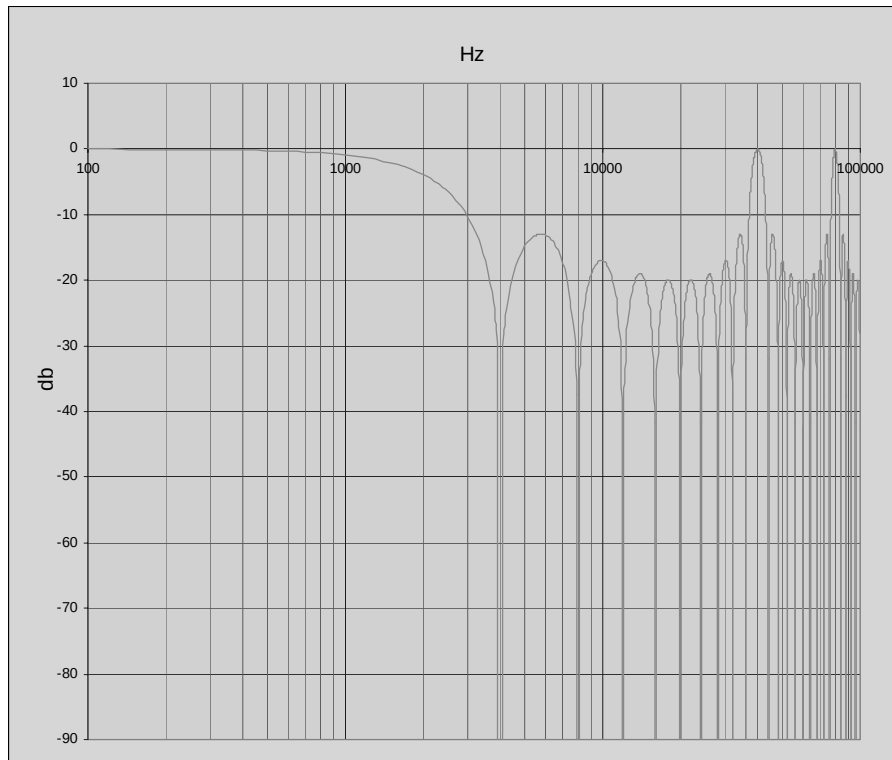


Figure 2. Moving Average or Comb Filter Gain Response

3.2 Forth-Order LPF

The forth-order LPF is constructed by cascading two identical second-order LPFs. These two filter stages are designed to be identical for simplicity only, the component values can be redesigned to be different from stage to stage. Figure 3 shows one possible form of a second-order filter.

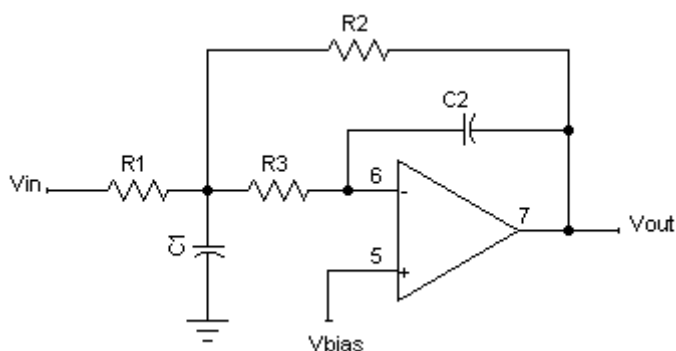


Figure 3. A Second Order Low-Pass Filter

If we select the component values such that $R1=R2=R3=R$, $C1= 3C/b$ and $C2 = bC/3$, the transfer function will be

$$H(s) = 1/(1+bRCs+(RCs)^2) \quad (4)$$

And the 3db frequency is

$$f_o = 1/(2\pi RC) \quad (5)$$

Component value selection does not need to be bound by the above rules; however, not doing so will make calculation more complicated and may necessitate software tools.

The selected values for the current design are $R1=R2=R3=47K$, $C1 = 0.01\mu F$ and $C2 = 100pF$. Both the response of the second-order and the combined forth-order filter are shown on Figure 4. The 40 kHz PWM carrier is reduced to almost zero as you can see there is more than 72 db or 12 bits of attenuation.

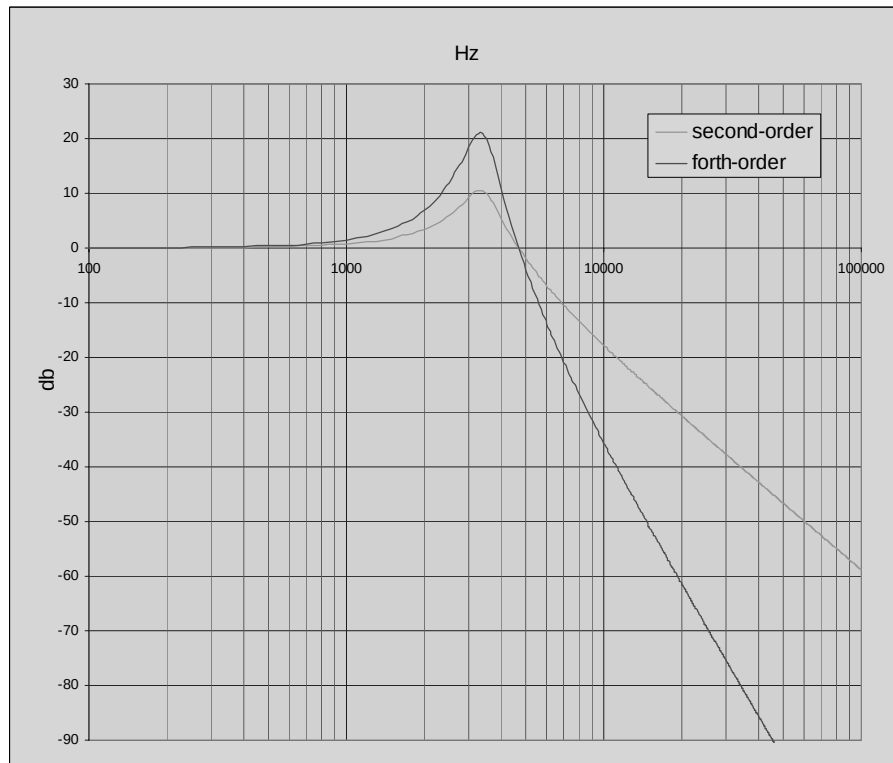


Figure 4. Output Analog LPF Gain Response

3.3 Combined Response

A noticeable characteristic of the current forth-order LPF response is that there is a peak around 3.5 kHz. This is to compensate for the roll off of the input moving average and output comb filter response. Figure 5 and Figure 6 show the combined response of the output section and the overall system. The system response shown does not include the input first-order LPF, as it has a very low impact over the 4 kHz frequency range we are most concerned with.

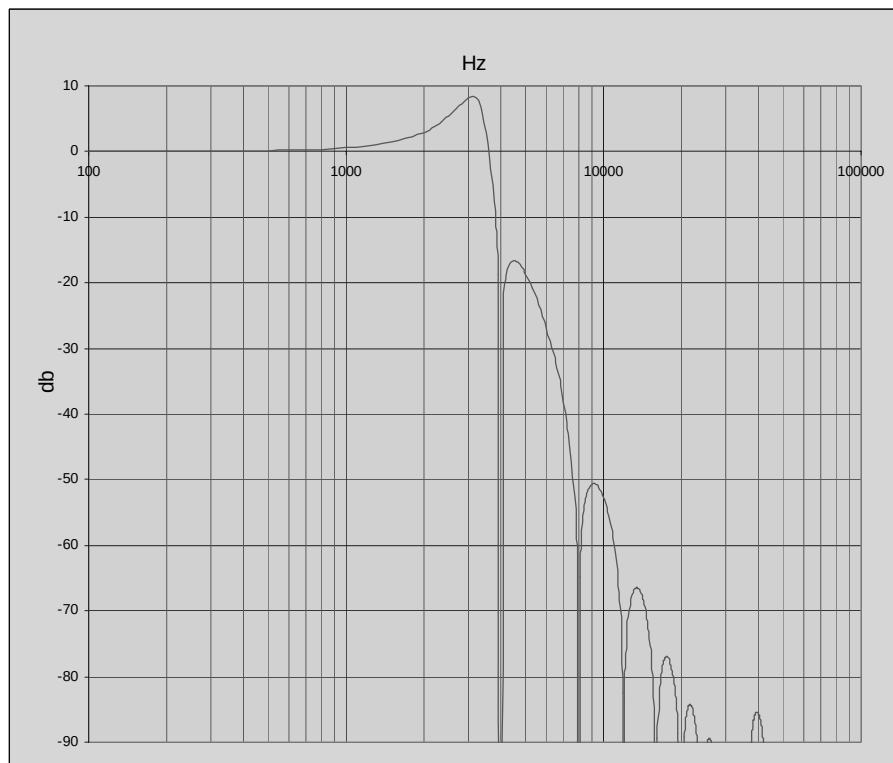


Figure 5. Output Section Gain Response

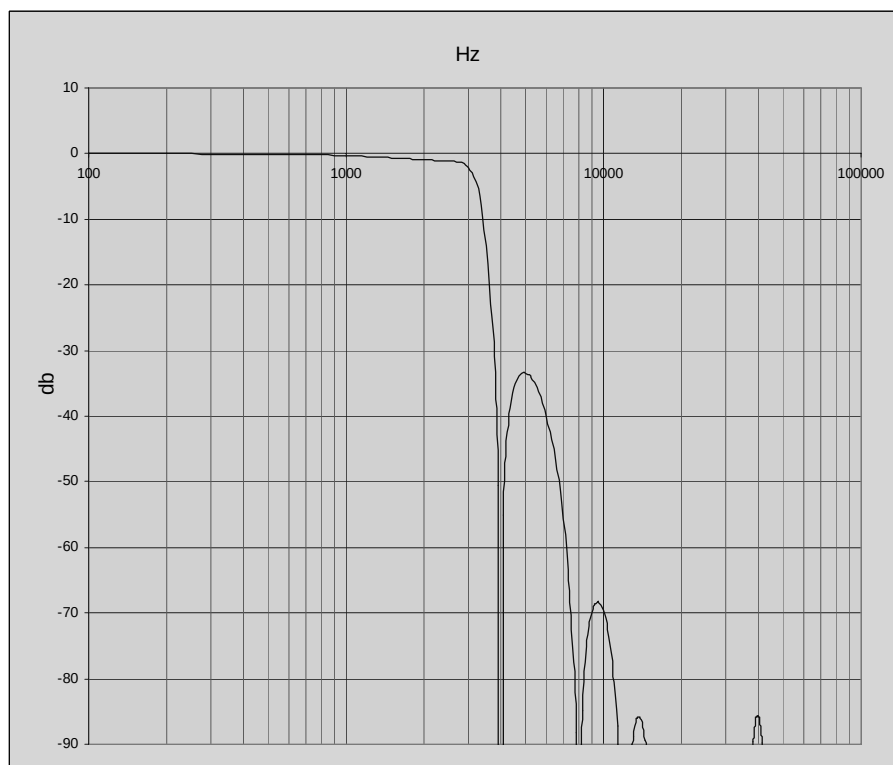


Figure 6. Overall System Gain Response

4 Oversampling

Oversampling is used to reduce the complexity of the post PWM filter. The effect of oversampling is to push the carrier frequency away from the signal frequency band.

Without oversampling, the carrier for an 8 ksps system is at 8 kHz. To filter this out adequately, a 12th order 72 db attenuation analog filter will be needed for data in 12 bit resolution. This will be a relatively much more expensive and space consuming circuit.

The downsampling and upsampling are more or less straight forward. For downsampling, the extra data of the moving average filter are dropped.

$$\text{i.e. } y[n] = x[k_d n] \quad (6)$$

where k_d is the downsampling ratio

The upsampling is achieved by filling out the undefined input data with the previous known sampled data and passing the results through a comb-filter .

$$\text{i.e. } y[k_u n] = x[n] \text{ and } y[k_u n + r] = x[n] \text{ for } r = 1, 2, \dots, k_u - 1 \quad (7)$$

where k_u is the upsampling ratio

We are using both $k_d = k_u = 5$ (i.e. 5 times oversampling) which is adequate for the filter design.

5 Dual PWM

In conventional PWM DAC, given a certain clock frequency, there is a sampling frequency versus resolution trade-off. To increase both without increasing the over all clock frequency (we want to keep the clock frequency as low as possible to minimize power consumption), we split the 12 bits DAC value into two 6 bits PWM channels. The channels are then added together in analog with the LSB channel scaled down by 64.

We can write any digital value as

$$x[n] = g \cdot x_2[n] + x_1[n] \quad (8)$$

If we convert $x_1[n]$ and $x_2[n]$ to analog and add them back together, we can get the analog value of $x[n]$.

$$x(nt) = g \cdot x_2(nt) + x_1(nt) \quad (9)$$

where $x(nt) = (V_{cc}/N) \cdot x[n]$

$$x_1(nt) = (V_{cc}/N) \cdot x_1[n]$$

$$x_2(nt) = (V_{cc}/N) \cdot x_2[n]$$

N = number of count for one PWM period

The selection of g should be a power of 2 such that $x_2[n]$ and $x_1[n]$ can be easily worked out. To combine the two PWM signals, a simple resistor network can be used.

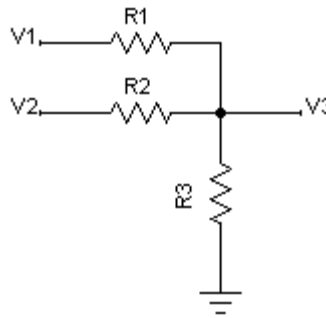


Figure 7. Combining Two PWM Outputs

In Figure 7, the input and output voltage is related by

$$V3 = (V1/R1 + V2/R2) / (1/R1 + 1/R2 + 1/R3)$$

If we put $R1 = g \cdot R2$, we get

$$V3 = p \cdot (V1 + g \cdot V2) \quad (10)$$

$$\text{where } p = 1 / (g \cdot R2 \cdot (1/R1 + 1/R2 + 1/R3))$$

Assuming the two PWM outputs have gone through an ideal low-pass filter such that $V1 = x_1(t)$ and $V2 = x_2(t)$, from equation (9) and (10), we get the analog output $V3 = p \cdot x(t)$.

In actual implementation, we do not really put two low-pass filters in front of the resistor network. Instead of doing this, we can put a single filter after the network. This produces the same result.

For 12 bits of digital data, an obvious choice of g is 64, which is a power of 2. So both $x_1[n]$ and $x_2[n]$ are 6-bits. $R1$ and $R2$ can be selected to 300 K and 4.7 K. The ratio is very close to 64 and the error is around 0.27%. This should be acceptable for a voice application. If higher accuracy is required, the resistors should be fine-tuned to the exact ratio. Another way to reduce error or distortion is to perform a calibration using the internal ADC after reset. A correction table can then be worked out and used later to digitally correct for the resistor errors.

6 Summary

We have described the design of a low cost codec based on the Texas Instruments' MSP430F13x mixed-signal microprocessor.

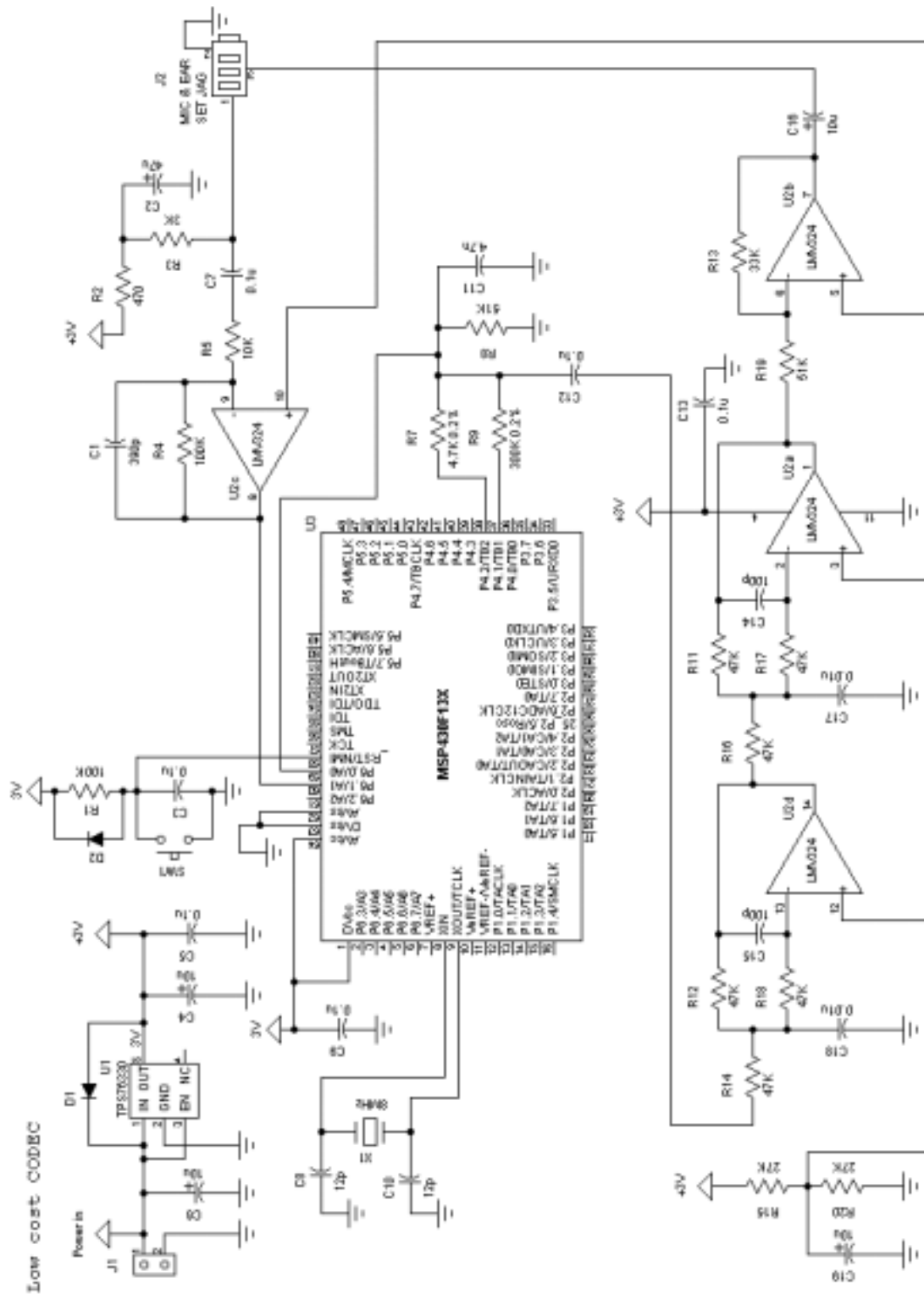
The PWM DAC design has been greatly simplified by using a dual PWM and oversampling technique. The various filters used in this design, both analog and digital have been described in detail.

The example presented here is only a loop-back for voice quality testing and it does not constitute an application. However, the codec will be an important component for various communication systems. For example, it can be used in a wireless headset for a mobile phone system. The input voice signal is converted by the ADC part of the codec to digital data. This data is compressed in ADPCM and transmitted out through RF. The data is then decompressed in the receiving end and is converted back to a voice signal with the DAC of the CODEC. Other applications that require ultra low power, low cost codec can make use of this design.

References

1. MSP430 Family, *Architecture Guide and Module Library User's Guide* (SLAUE10)
2. MSP430x1xx Family *User's Guide* (SLAU049)
3. MSP430x13x/14x *data sheet* (SLAS272)
4. *Using PWM Timer_B as a DAC* (SLAA116)
5. *A Simple Approach to Digital Signal Processing* (SPRDE01A)

Appendix A. Schematic



Appendix B. Software Listing

```

;*****
; File:          codec_loopback.s43
;
; Description:    This is a demo code for the low cost speech codec using internal
loopback.
;
; Revisions: 1.0    13 Feb 2001
;
; Copyright: Texas Instruments Hong Kong Ltd
;
; Note to users:
;       This code is given in an as is basis. Texas Instruments does not
;       take responsibility on the maintenance of this code.
;
;   Kes Tam
;   Texas Instruments Hong Kong
;
;   Comments:
;
;   2001/2/13
;
;*****;-----
-----
; Compilation control
;-----
DIRECT_IN_OUT equ    0

COMB_FILTER  equ    1
COMB2        equ    0
ADC_CUTOFF_4K equ    1
DAC_CUTOFF_4K equ    1          ;valid for COMB_FILTER = 1 and COMB2 = 0
EIGHT_STATES equ    0

;-----
; include available peripheral port addresses
;-----
#include      "msp430x13x.h"

```

```

;-----
; Pin assignment
;-----
;
;           MSP430F13x/F14x
;
;  .-----
;  /|\ |
;  | |           XIN |---
;  ---| RST           | 8MHz
;  | |           XOUT |---
;  | |           |
;  | |           P6.1/A1 |<--- Mic in
;  | |           |
;  | |           P4.1/TB1 |--- 4.7K --+
;  | |           |           |
;  | |           P4.2/TB2 |--- 300K --+--- Ear set out
;  | |           |
;
;
;-----

#define P1OUT_DEFAULT 00000000b,&P1OUT
#define P2OUT_DEFAULT 00000000b,&P2OUT
#define P3OUT_DEFAULT 00000000b,&P3OUT
#define P4OUT_DEFAULT 00000000b,&P4OUT
#define P5OUT_DEFAULT 00000000b,&P5OUT
#define P6OUT_DEFAULT 00000000b,&P6OUT

#define P1DIR_DEFAULT 00000000b,&P1DIR
#define P2DIR_DEFAULT 00000000b,&P2DIR
#define P3DIR_DEFAULT 00000000b,&P3DIR
#define P4DIR_DEFAULT 00000110b,&P4DIR
#define P5DIR_DEFAULT 00000000b,&P5DIR
#define P6DIR_DEFAULT 00000000b,&P6DIR

#define P1SEL_DEFAULT 00000000b,&P1SEL
#define P2SEL_DEFAULT 00000000b,&P2SEL
#define P3SEL_DEFAULT 00000000b,&P3SEL

```

```
#define      P4SEL_DEFAULT 00000110b,&P4SEL
#define      P5SEL_DEFAULT 00000000b,&P5SEL
#define      P6SEL_DEFAULT 00000011b,&P6SEL

;-----
; define memory start
;-----
RAM_orig     EQU     00200h ; assume 256 bytes
FLASH_orig   EQU     0F000h ; assume 4K size

;-----
; constants for rf speed
;-----
CLOCK_RATE   equ     8000           ;in KHz
SAMPLING_RATE equ     80           ;sampling in KHz
BIT_TIME     equ     (CLOCK_RATE/SAMPLING_RATE)

;-----
; PWM output DC offset
;-----
V_MID_OFFSET equ     (1000*BIT_TIME/917/2)           ;define offset to adjust the PWM DAC dc offset
to mid of Vcc
                                           ;offset_count=0.5/(51K/(4.7K//300K+51K))*BIT_TIME

PWM1_MAX     equ     63             ;max value of 6 bits
PWM2_MAX     equ     ((63*5*5*3)/(4*4*4))           ;max value of 6 bits plus gain
                                           ;5=ADC input filter gain
                                           ;5=DAC output filter gain
                                           ;4,4,3/4 = internal multipliers

PWM1_OFFSET  equ     (V_MID_OFFSET-(PWM1_MAX/2))
PWM2_OFFSET  equ     (V_MID_OFFSET-(PWM2_MAX/2))

        if      ((BIT_TIME < (V_MID_OFFSET+PWM2_MAX/2)) | (BIT_TIME < V_MID_OFFSET+PWM1_MAX/2))
PWM overflow error           ;this line pretends to give compilation error
                           ;as PWM timer overflow
        endif

;-----
```

```

; other constants
;-----

;-----
; define register
;-----
#define      W1_reg      R15
#define      W2_reg      R14
#define      state_ctrl  R4
    if      ADC_CUTOFF_4K
#define      last_adc_sum r6
    endif
#define      y_out      r7
    if      COMB_FILTER
#define      x_in      r9
        if      COMB2
#define      x_delta1   r10
#define      x_delta2   r11
#define      last_y     r12
        else
            if      DAC_CUTOFF_4K
#define      last_x_in  r8
            endif
#define      x_delta    r10
        endif
    else
#define      x1         r8
#define      x2         r9
    endif

;@@@
#define      in_ptr      r12
#define      out_ptr     r13

;-----
; User defined RAM location
;-----

RSEG    UDATA0

```

```

dummy ds      0

PWM_BUFFER_SIZE      equ      8              ;@@@ must be in power of 2
pwm_buffer ds      PWM_BUFFER_SIZE      ;@@@ must be on top
drop_counter ds      2              ;@@@
drop_ctrl ds      2              ;@@@

;-----
; Program Start address
;-----

NAME    pwm_loopback

        RSEG    CODE
main
RESET      mov      #(RAM_orig+100h),SP      ; Initialize stackpointer
SetupWDT    mov      #(WDTPW+WDTHOLD),&WDTCTL ; Stop WDT

;*****
;
; SETUP CLOCK
;*****
SetupBC     bis.b   #XTS,&BCSCTL1      ; turn on high frequency crystal
wait_HF_oscillator      ; This loop will wait till
        BIC.b   #0FIFG,&IFG1      ; high frequency crystal is stable
        BIT.b   #0FIFG,&IFG1
        JnZ     wait_HF_oscillator
        call    #delay            ; wait for crystal stable

        BIC.b   #0FIFG,&IFG1      ; Reset osc. fault flag again

                                ; Then set MCLK same as LFXTCL
                                ; and SMCLK = LFXTCL/4
        mov.b   #(DIVS1+SELS+SELM0+SELM1),&BCSCTL2

;*****
;
; SETUP PORT
;*****
        mov.b   #P4OUT_DEFAULT

```

```

mov.b #P4DIR_DEFAULT
mov.b #P4SEL_DEFAULT

mov.b #P6OUT_DEFAULT
mov.b #P6DIR_DEFAULT
mov.b #P6SEL_DEFAULT

;*****
;
; setup ADC12
;*****
and    #0fffdh,&ADC12CTL0 ;selection of reference and input
mov.b  #00001h,&ADC12MCTL0 ;multiple conversion of a single channel
mov.b  #00001h,&ADC12MCTL1
mov.b  #00001h,&ADC12MCTL2
if     EIGHT_STATES
mov.b  #00081h,&ADC12MCTL3
else
mov.b  #00001h,&ADC12MCTL3
endif
if     0
mov.b  #00001h,&ADC12MCTL4
mov.b  #00001h,&ADC12MCTL5
mov.b  #00001h,&ADC12MCTL6
mov.b  #00001h,&ADC12MCTL7
mov.b  #00001h,&ADC12MCTL8
mov.b  #00081h,&ADC12MCTL9
else
mov.b  #00081h,&ADC12MCTL4
endif
if     DIRECT_IN_OUT
mov    #00904h,&ADC12CTL1 ;repeat single channel
else
mov    #00906h,&ADC12CTL1 ;first conv. result is stored in ADC12MEM0
                                ;ADC12SC bit triggers Sample&Hold
                                ;ISSH trigger is inverted
                                ;sample pulse is generated by Sampling Timer
                                ;Clock Source: TIMER_B OUT 0
                                ;Clock divider: 1
                                ;repeated sequence of conversion

```

```

        endif
;      mov    #02210h,&ADC12CTL0 ;select 2*4*ADC12CLK for sampling time
      mov    #00010h,&ADC12CTL0 ;Sample&Hold Time 0
                                   ;Sample&Hold Time 1
                                   ;Multiple Sample&Hold OFF
                                   ;reference voltage is off
                                   ;ADC12 module is switched on
                                   ;Interrupt at the end of every ADC conversion

      mov    #00000h,&ADC12IE    ;disable interrupt
                                   ;everytime each buffer memory has ADC data
                                   ;clear automatically

      bis    #00002h,&ADC12CTL0 ;enable conversion

;*****
;  set up timer_B for PWM
;*****
      mov    #(BIT_TIME-1),&TBCCR0
      mov    #00000h,&TBCCR1
      mov    #(BIT_TIME/2),&TBCCR2
      mov    #00090h,&TBCCTL0    ;set outmode 4 for toggle, Interrupt enabled
;      mov    #00080h,&TBCCTL0    ;set outmode 4 for toggle
      mov    #002e0h,&TBCCTL1    ;set outmode 7 PWM reset/set
                                   ;load new CCR1 when TBR = 0

      mov    #002e0h,&TBCCTL2    ;set outmode 7 PWM reset/set
                                   ;load new CCR2 when TBR = 0

      mov    #00114h,&TBCTL      ;start TIMER_B up mode, select AMCLK as input clock

;*****

      clr    state_ctrl

      if     ADC_CUTOFF_4K
      clr    last_adc_sum
      endif
      if     COMB_FILTER
      clr    x_in
      clr    y_out

```

```

        if      COMB2
        clr     x_delta1
        clr     x_delta2
        clr     last_y
        else
                if      DAC_CUTOFF_4K
        clr     last_x_in
                endif
        clr     x_delta
        endif
    else
        clr     y_out
    endif

    mov     #pwm_buffer,in_ptr    ;@@@
    mov     #(pwm_buffer+(PWM_BUFFER_SIZE>>1)),out_ptr    ;@@@
    mov     #40000,drop_counter
    clr     drop_ctrl            ;@@@

    eint

main_loop
    bis     #CPUOFF,SR
    jmp     main_loop

;*****
; subroutine :      delay
; function : software delay
;*****
delay
    push    #0FFFFh            ; Delay for crystal start
L1    dec     0(SP)
    jnz     L1
    incd    SP
    ret

;***** interrupt service routines *****

```

```

;*****
; subroutine :      TBCCR0_isr
; function : Timer B CCR0 interrupt service routine
;*****

TBCCR0_isr
    if    DIRECT_IN_OUT
    mov    &ADC12MEM0,W2_reg
    mov    W2_reg,W1_reg
    and    #03fh,W1_reg
    add    #23,W1_reg
    rla    W2_reg                ;right shift 6 bits
    rla    W2_reg
    swpb   W2_reg
    and    #0ffh,W2_reg
    add    #23,W2_reg
    mov    W2_reg,&TBCCR2
    mov    W1_reg,&TBCCR1
    reti
    else

    incd   state_ctrl
    add    state_ctrl,pc
    jmp    ST0
        if    !EIGHT_STATES
    jmp    ST1
    jmp    ST2
        endif
    jmp    ST3
    jmp    ST4
    jmp    ST5
    jmp    ST6
    jmp    ST7
    jmp    ST8
    jmp    ST9

ST0
    mov    W2_reg,&TBCCR2
    mov    W1_reg,&TBCCR1

```

```

    if      1
    mov     #ADC12MEM0,W1_reg
    mov     @W1_reg+,W2_reg
    add     @W1_reg+,W2_reg
    add     @W1_reg+,W2_reg
    add     @W1_reg+,W2_reg

    if      EIGHT_STATES
    mov     W2_reg,W1_reg           ;mul 5/4
    rra     W1_reg
    rra     W1_reg
    add     W1_reg,W2_reg
    else
    add     @W1_reg+,W2_reg
    endif
    else
    inc     r5
    inc     r5
    bic     #0fff1h,r5
    mov     SIN_TABLE(r5),W2_reg
    endif

    if      ADC_CUTOFF_4K
    mov     W2_reg,W1_reg
    add     last_adc_sum,W2_reg
    rrc     W2_reg                 ;unsigned shift
    mov     W1_reg,last_adc_sum
    endif

    rra     W2_reg
    rra     W2_reg

    if      COMB_FILTER

        if      COMB2
        rra     W2_reg             ;avoid overflow as gain=25 for 2nd order comb filter
                                    ;however, this reduce the resolution to 11 bits

```

```

        mov    y_out,W1_reg                ;this will be last_y

        rla    y_out                      ;calculate current y
        sub    last_y,y_out
        add    x_delta2,y_out
        sub    x_delta1,y_out

        mov    W1_reg,last_y              ;update last_y

        mov    x_delta2,x_delta1          ;get new x_delta1
        mov    W2_reg,x_delta2
        sub    x_in,x_delta2              ;get new x_delta2
        mov    W2_reg,x_in                ;get current x

        else

        add    x_delta,y_out              ;get current y

        if     DAC_CUTOFF_4K

;        rra    W2_reg                      ;avoid overflow and compensate extra gain of 2
        mov    W2_reg,x_delta
        sub    last_x_in,x_delta          ;get current x_delta
        mov    x_in,last_x_in

        else

        mov    W2_reg,x_delta
        sub    x_in,x_delta                ;get current x_delta
        endif

        mov    W2_reg,x_in                ;get current x
        endif

        mov    y_out,W2_reg

        else                                ;moving avg
        mov    x2,x1
        mov    W2_reg,x2
        mov    x1,W2_reg
        rla    W2_reg                      ;mul 5
        rla    W2_reg
        if     !EIGHT_STATES

```

```

    add    x1,w2_reg
    endif

    mov     w2_reg,y_out
    endif
    jmp     TBCCR0_isr_exit

```

ST2

ST4

ST6

ST8

```

    mov     w2_reg,&TBCCR2
    mov     w1_reg,&TBCCR1

    if      COMB_FILTER

        if      COMB2
    mov     y_out,w1_reg                ;this will be last_y

    rla     y_out                      ;calculate current y
    sub     last_y,y_out
    add     x_delta2,y_out
    sub     x_delta1,y_out

    mov     w1_reg,last_y              ;update last_y

        else

    add     x_delta,y_out              ;update current y

        endif

    mov     y_out,w2_reg

    else
    sub     x1,y_out
    add     x2,y_out
    mov     y_out,w2_reg
    endif

```

```

        jmp    TBCCR0_isr_exit

ST9     mov    #-2,state_ctrl
ST1
ST3
ST5
ST7
        mov    W2_reg,0(in_ptr)          ;@@@
        incd   in_ptr                    ;@@@

        mov    @out_ptr+,W2_reg
        dec    drop_counter
        jnz    no_drop
        mov    #100,drop_counter
        decd   in_ptr
        decd   out_ptr
no_drop
        bic    #PWM_BUFFER_SIZE,in_ptr    ;@@@
        bic    #PWM_BUFFER_SIZE,out_ptr   ;@@@

        if     (COMB_FILTER & (COMB2 | DAC_CUTOFF_4K))
        clrc
        rrc    W2_reg                    ;unsigned shift
        endif

        rra    W2_reg
        rra    W2_reg
        if     EIGHT_STATES
        mov    W2_reg,W1_reg
        rra    W1_reg
        rra    W1_reg
        add    W1_reg,W2_reg
        endif

        if     (COMB_FILTER & COMB2)
        rra    W2_reg
                if     1                ;mul 5/4
        mov    W2_reg,W1_reg
        rra    W1_reg

```

```

    rra    W1_reg
    add    W1_reg,W2_reg

        endif

    else
    mov     W2_reg,W1_reg        ;mul 3/4
    rra     W1_reg
    rra     W1_reg
    sub     W1_reg,W2_reg
    endif

    mov     W2_reg,W1_reg
    and     #03fh,W1_reg
    rla     W2_reg                ;right shift 6 bits
    rla     W2_reg

    swpb    W2_reg
    and     #0ffh,W2_reg        ;more than 6 bits required
    add     #PWM2_OFFSET,W2_reg
    add     #PWM1_OFFSET,W1_reg

```

```
TBCCR0_isr_exit
```

```
    reti
```

```
endif
```

```
;-----
```

```
SIN_TABLE
```

```

    dw      (2048+355)*5
    dw      (2048+1674)*5
    dw      (2048+2012)*5
    dw      (2048+1172)*5
    dw      (2048-355)*5
    dw      (2048-1674)*5
    dw      (2048-2012)*5
    dw      (2048-1172)*5

```

```
;-----
```

```
    RSEG    INTVEC ; MSP430x13x/14x Interrupt vectors
```

```
;-----
```

```

        DW      RESET          ; no source
        DW      RESET          ; P2.x

        DW      RESET          ; USART1 TX
        DW      RESET          ; USART1 RX
        DW      RESET          ; P1.x
        DW      RESET          ; Timer_A1, Timer_A2, TAR overflow
        DW      RESET          ; Timer_A0
        DW      RESET          ; ADC12
        DW      RESET          ; USART0 TX
        DW      RESET          ; USART0 RX
        DW      RESET          ; Watchdog/Timer, Timer mode
        DW      RESET          ; Comparator_A
        DW      RESET          ; Timer_B1 to Timer_B6, TBR overflow
        DW      TBCCR0_isr      ; Timer_B0
        DW      RESET          ; NMI, Osc. fault, Flash access violation
        DW      RESET          ; POR, ext. Reset, Watchdog
;-----
        END      main

```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265