Z4229 - Continuous Variable Slope Delta modulation vocoder implementation using the Z8Encore micro controller

This project is a proof of concept that the Z8Encore ADC and PWM peripherals can be used for voice processing with good results.
Delta modulation has evolved into a simple, efficient method of digitizing voice for secure, reliable communications and for voice I/O in data processing.
Companded PCM, for telephone quality transmission, requires about 64K bits/sec data rate per channel.
CVSD produces at 16K bits/sec an acceptable reconstructed voice but the sound is reminiscent of a damaged loudspeaker.
Because of the simplicity of the algorithm is easy to implement CVSD on a small CPU like Z8 and get positive results. Implementing the digital version of CVSD has some advantages over the analog counterpart: no analog integrator, the time constants are set by the clock frequency and do not drift with time or temperature.
The integrators are initialized at 0 each time the process starts.
The Delta encoder / decoder is model is the one proposed by Greefkes and Riemens in 1970 and known as CVSD.

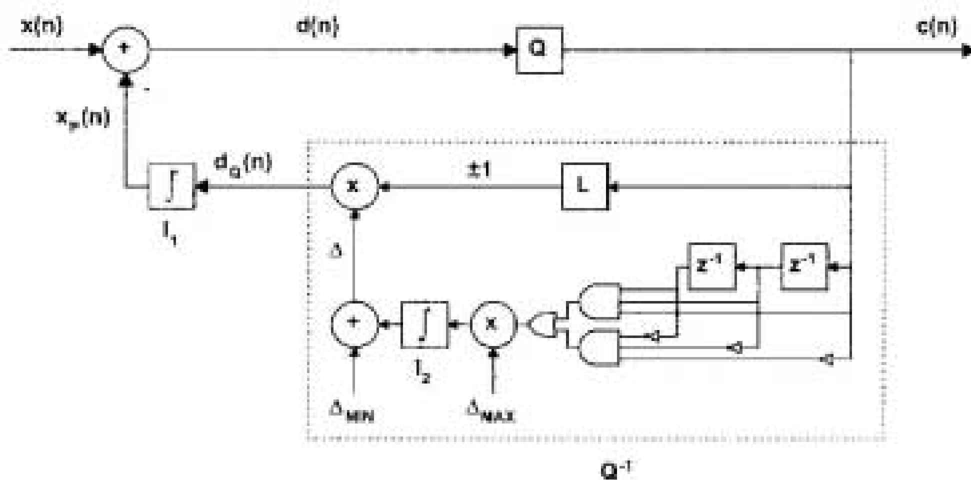


Figure 1: CVSD Encoder

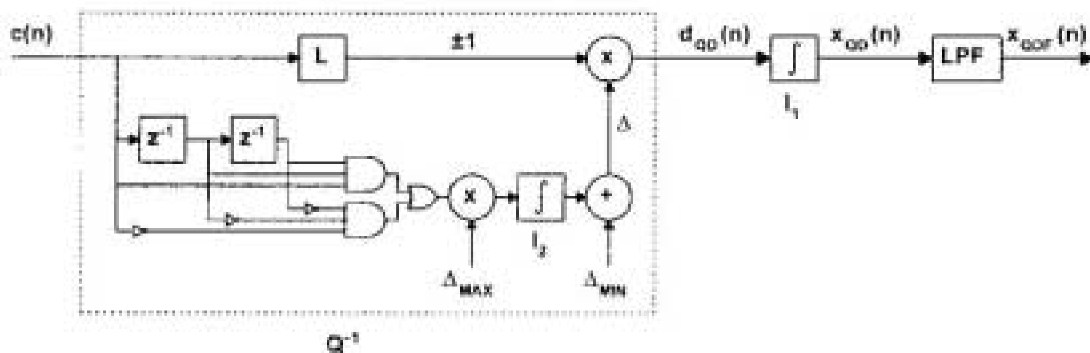


Figure 2: CVSD Decoder

The I1 integrator must have a constant of 1mS and the I2 integrator also known as the syllabic integrator must have a constant of 4-10mS.
If the signal has a high amplitude and high frequency then the three 1's or three 0's will be obtained at the output and the I2 integrator will start to modify the slope of the I1 in order to adapt to the necessary amplitude. Fortunately the human does not contain to many components with high amplitude an d high frequency at the same time.

For the actual implementation a sampling frequency of 18khz was used in order to derive it directly from the 18.432Mhx system clock.
The amplitude values are limited to 8 bits in this example. The values for the signal will be in the +/- 128 range. Choosing maximum amplitude of 108 instead of 128 we get a step of 6 for a 1mS integrator constant (108/6=18 X Sampling Period = 1mS) and a amplitude overload margin of 18%(108/128).
For the second integrator a 4mS constant is necessary and this will be a 1.5 step – because of integer math used a step of two was selected.

Next step was to use some software model to validate the design. An Excel program was a good choice in order to display the data and do some simple processing. Please see the d18000.xls file for complete details and to get more simulation results. A sinusoidal signal is produced; the signal is encoded and then decoded. The results are presented on a plot for the first 100 samples and the last 100 samples of 300 in order to see the effect of slow starting for I2, some amplitude issues and any other stability problems.
It is possible to change the values for the signal amplitude/frequency and the constants for the two integrators.
The plots presented are the result for 18000Hz Sampling Frequency, a 500Hz test signal with a 100 amplitude.

First 100 samples

122
72
22
-28
-78
-128

1 11 21 31 41 51 61 71 81 91

- Input signal
- Delta Output
- Decoded signal

Last 100 Samples

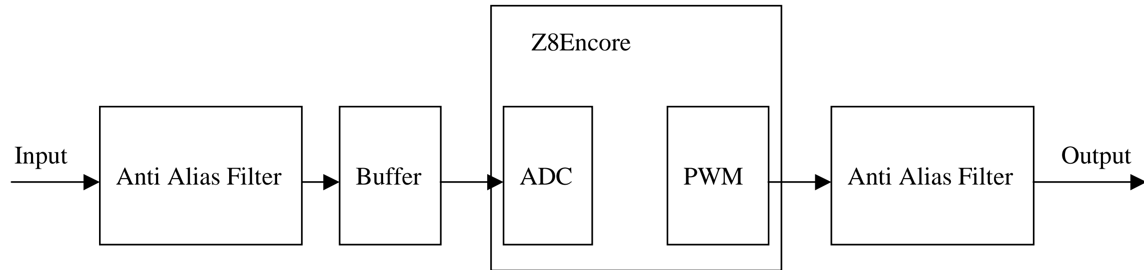
122
72
22
-28
-78
-128

1 11 21 31 41 51 61 71 81 91 101

- Input Signal
- Delta Output
- Decoded Signal

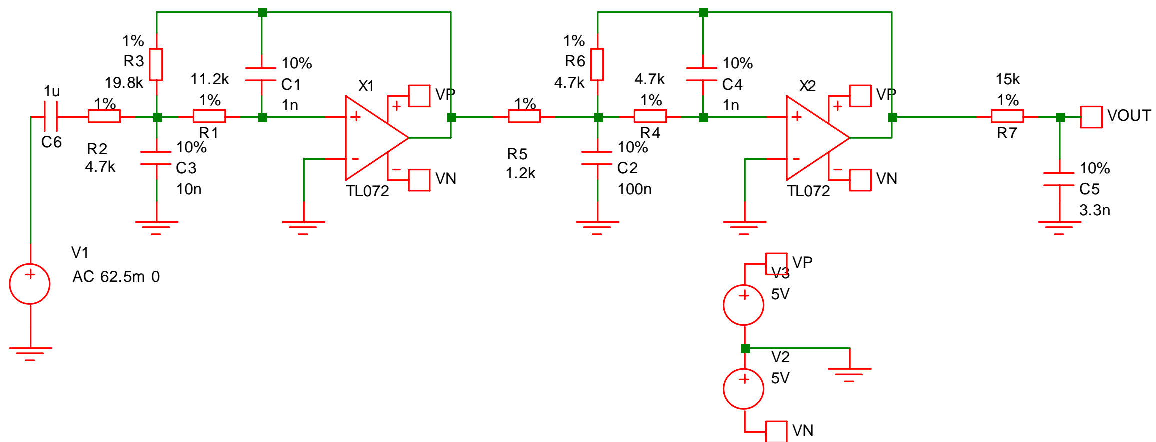
Hardware

The hardware consists of the Anti Alias Filters and an impedance matching circuit for the ADC. The Z8Encore development kit is used to support the Z8Encore CPU.

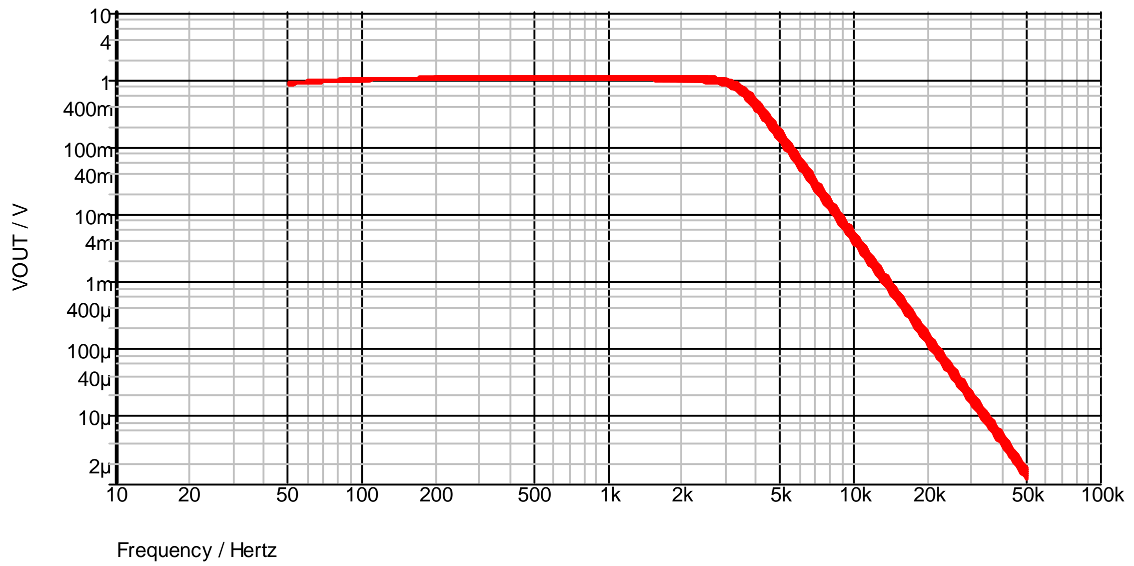


Anti Alias Filter

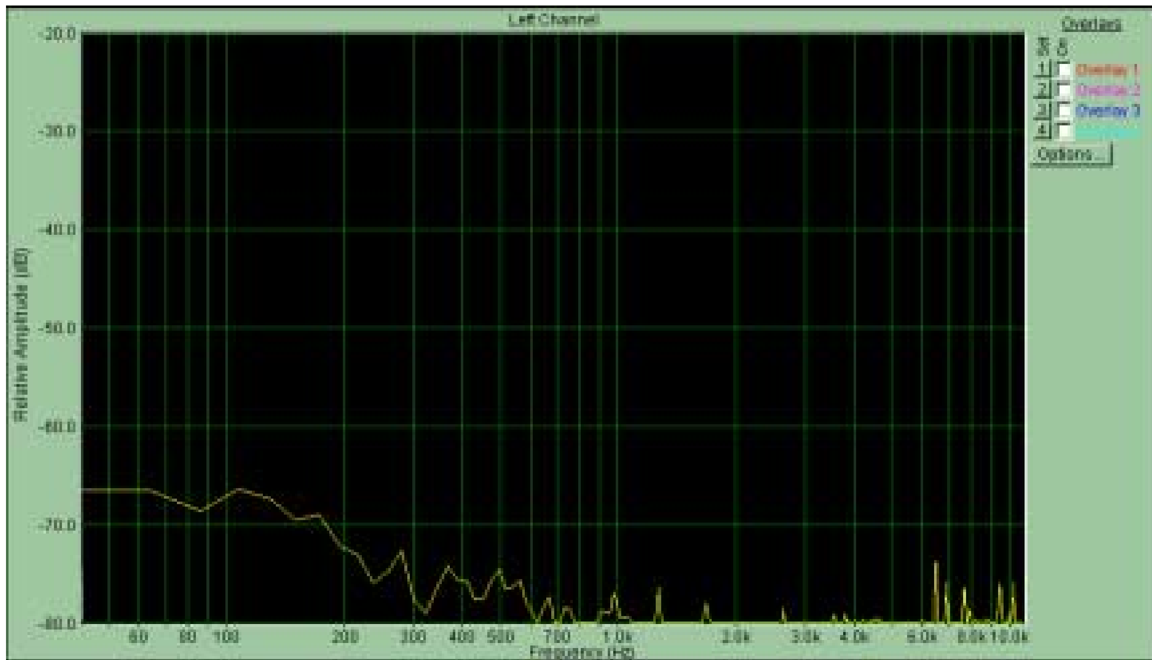
Parameters:
Cutoff Frequency: 3.4Khz (-3dB)
Filter Type: Butterworth (no ripple)
Filter Order: 5
Multiple Feedback Topology Filter
Gain = 16
TL072 and MCP6021 with single 5Volt Supply used for simulation



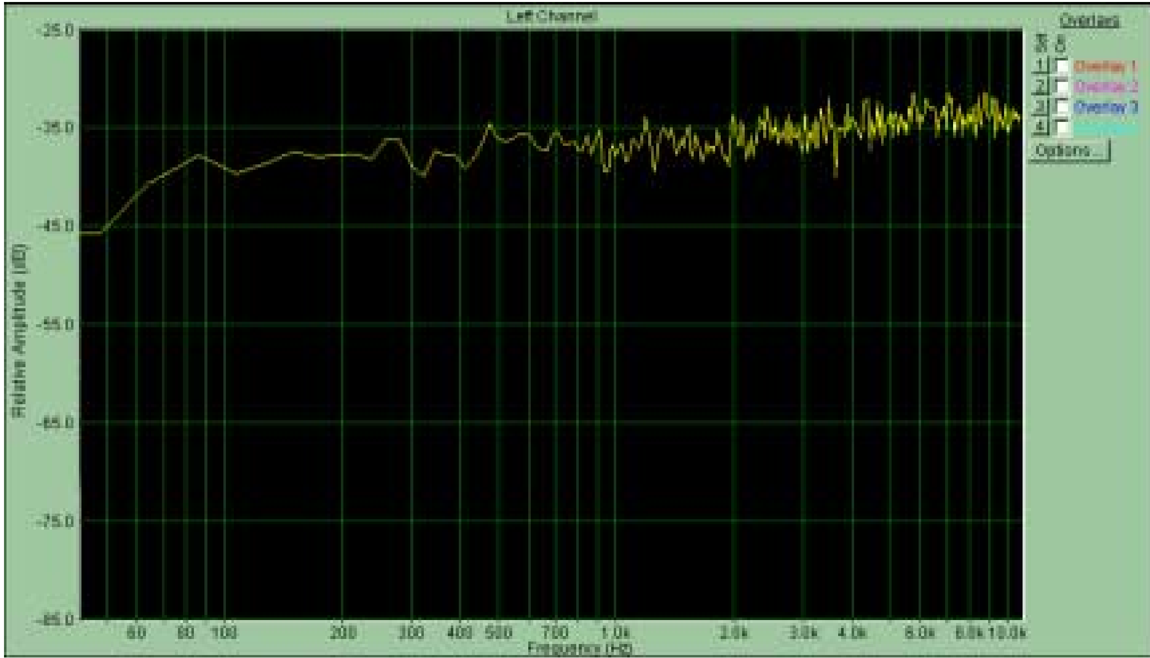
Schematic Diagram



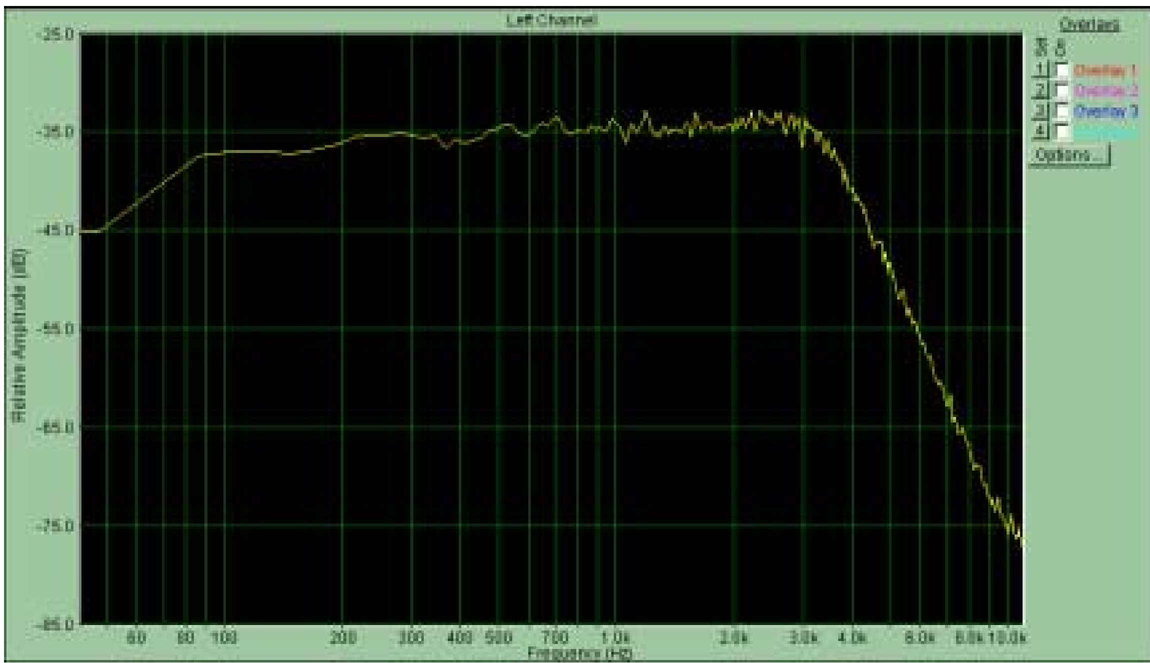
Monte Carlo Simulation with R tolerance set to 1% and C tolerance set to 10%
40dB Attenuation at ~8Khz



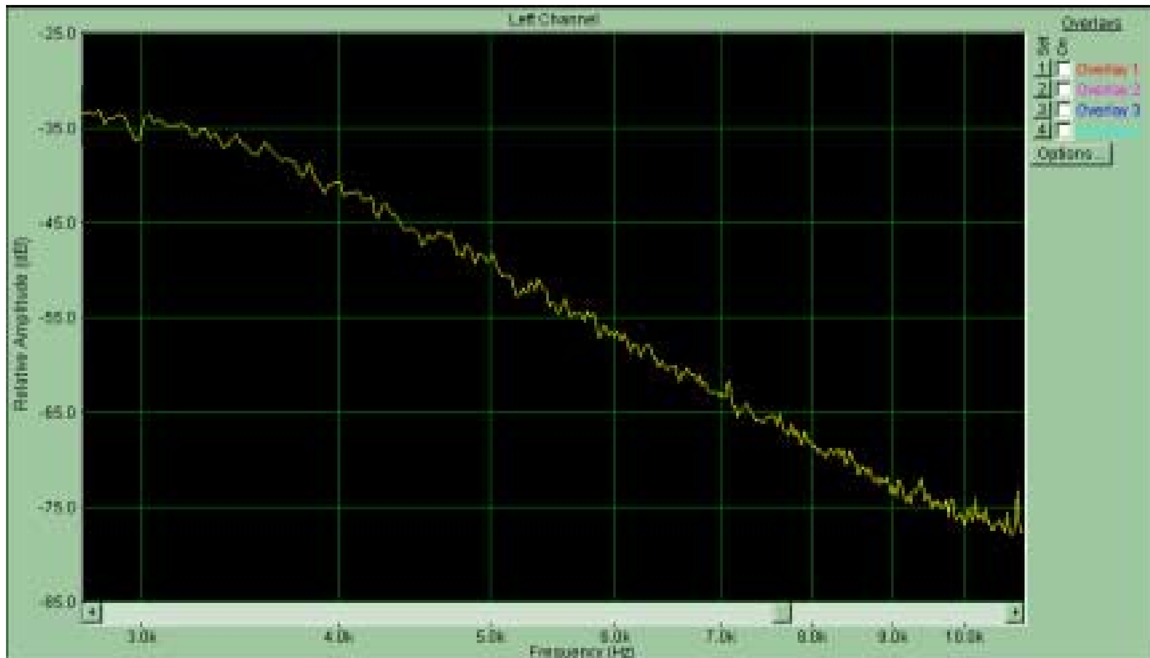
Mic input noise



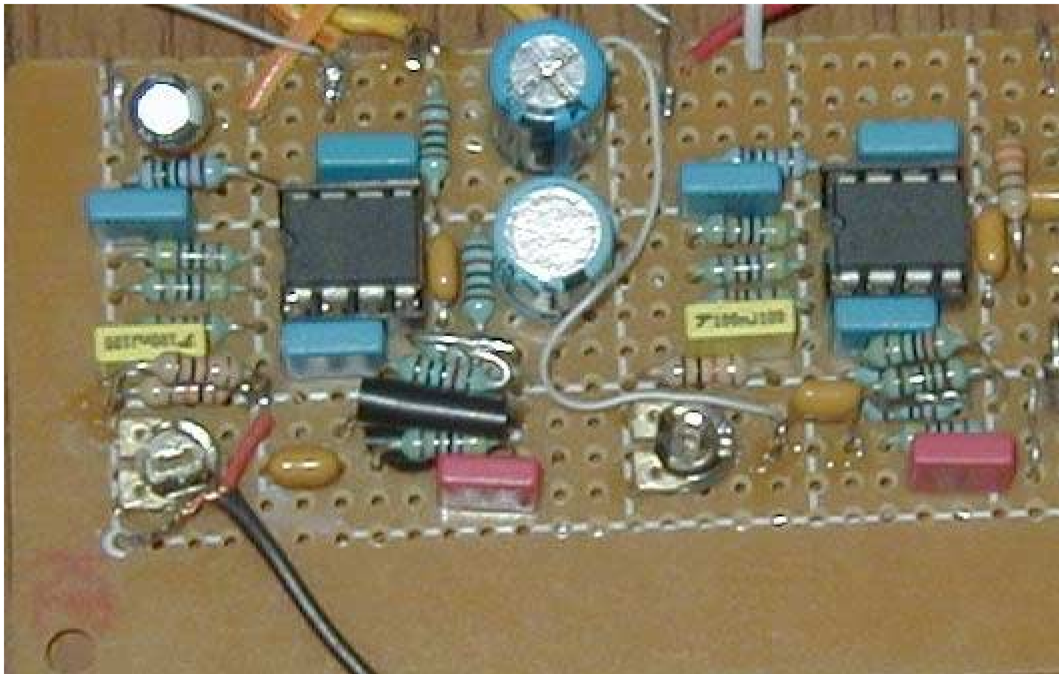
Reference Pink Noise Level – without the filter



Filter output - 40dB Attenuation at 10khz



Detail - Low Pass Filter



Detail – Anti Alias Filter implementation using MCP6022 dual opamps.

Buffer Circuit for ADC

The buffer circuit is built around an OPA348 from Texas Instruments. A classical non-inverting buffer configuration is used. The opamp is powered with 3.3V like the CPU.

This stage was necessary because of the low input impedance of the ADC. The SMD IC was kind of difficult to use but the rail-to-rail performance at 3.3V power supply is great. In the attached Picture we can see the small opamp and a 100-ohm resistor in series with the output in order to prevent the situation when the ADC input can be configured like output.

Table 106. Analog-to-Digital Converter Electrical Characteristics and Timing

Symbol	Parameter	V_{DD} = 3.0 - 3.6V T_A = -40°C to 105°C			Units	Conditions
		Minimum	Typical	Maximum		
	Resolution	–	10	–	bits	External V_{REF} = 3.0V; R_S <= 3.0kΩ
	Differential Nonlinearity (DNL)	-1.0	–	1.0	LSB	External V_{REF} = 3.0V; R_S <= 3.0kΩ
	Integral Nonlinearity (INL)	-3.0	–	3.0	LSB	External V_{REF} = 3.0V; R_S <= 3.0kΩ
	DC Offset Error	-35	–	25	mV	80-pin QFP and 64-pin LQFP packages.

[1] Analog source impedance affects the ADC offset voltage (because of pin leakage) and input settling time.

Symbol	Parameter	V_{DD} = 3.0 - 3.6V T_A = -40°C to 105°C			Units	Conditions
		Minimum	Typical	Maximum		
	DC Offset Error	-50	–	25	mV	40-pin PDIP, 44-pin LQFP, 44-pin PLCC, and 68-pin PLCC packages.
V_{REF}	Internal Reference Voltage	–	2.0	–	V	
	Single-Shot Conversion Time	–	5129	–	cycles	System clock cycles
	Continuous Conversion Time	–	256	–	cycles	System clock cycles
	Sampling Rate	System Clock / 256			Hz	
	Signal Input Bandwidth	–	–	3.5	kHz	
R_S	Analog Source Impedance	–	–	10[1]	kΩ	
Zin	Input Impedance		150		kΩ	20MHz system clock. Input impedance increases with lower system clock frequency.
V_{REF}	External Reference Voltage			AVDD	V	AVDD <= VDD. When using an external reference voltage, decoupling capacitance should be placed from VREF to AVSS.

[1] Analog source impedance affects the ADC offset voltage (because of pin leakage) and input settling time.

Software

The software is interrupt driven and is using the ADC and PWM to acquire and output the signal.
The PWM module is using Timer1. Because the PWM unit does not have buffer registers a small trick is used. A 9 bit PWM is used but the values are only 8 bit. The 8 bit values are centered on ½ of the 9 bit maximum value (512). Therefore the PWM value will be greater than 128 at any moment. Loading the value immediately when the reload interrupt occur (in less than 128 system clock cycles) will keep the PWM on the safe side.

```
	PCADDR = 0x01;			// Port C data direction
	PCCTL  &= 0xF8;			// PC1 = Timer1_out is output. PC0=out,PC2=out, Rest all 
inputs.
	PCADDR = 0x02;			// Port C alternate function,
	PCCTL  |= 0x02;			 // Enabled Timer1(PWM output)
	PCADDR = 0x03;			// Port C Open Drain control
	PCCTL  &= 0xFD;			// Timer1(PWM output) NOT open drain
```

```
PCADDR = 0x04;                  // Port C HIGH DRIVE control
PCCTL  |= 0x02;                 // Timer1(PWM output) High Drive Enable
PCADDR = 0x00;                  // protect port control


/****************************************************************
* Initialize Timer 1 in continuous PWM mode with prescale = 1
* Initialize the PWM width  0x0100 - 1/2 scale.
* Initialize the PWM period to 27.77us / 36khz.
****************************************************************/
T1CTL   = 0x43;           // PWM mode, Prescale 1, TPOL=0, not enabled.
 T1PWMH  = 0x01;          // PWM High
T1PWML  = 0x00;           // PWM Low       1/2 scale

T1CPH   = 0x02;           // RELOAD Hi/ Low = 0x0200 with a prescaler of 1
 T1CPL   = 0x00;          // results 36KHz PWM cycle at 18.432MHz sysclk

T1CTL  |= 0x80;           // Enable timer PWM
```

The ADC is used in continuous mode. A new conversion will end every 256-clock cycles.

```
PBADDR = 0x02;                  // Port B alternate function,
PBCTL  |= 0x01;     // Enabled PB0=ANA0=ADC INPUT 0
        PBADDR = 0x03;                  // Port C OPEN DRAIN control
        PBCTL  &= 0xFE;                 // PB0=ANA0 NOT open drain
        PBADDR = 0x04;                  // Port C HIGH DRIVE control
        PBCTL  &= 0xFE;                 // PB0=ANA0 NOT High Drive
        PBADDR = 0x0;

        ADCCTL = 0xB0;                  // CEN=1, 0, VRED=1, CONT=1, ANAIN=0000
        // CEN will be 0 at first complete conversion result 5129+40 system clock cycles

      while (ADCCTL & 0x80 ) //wait the end of the first A/D conversion
```

The PWM interrupt will occur 36000 times per second. Because the Sampling Frequency is only 18.000Hz only one of two interrupts will be used to change the PWM value. Port C out 0 is used to output the Sample Frequency; Port C output 1 is the PWM output.

```
#pragma interrupt
void interrupt dac_isr(void)
{
        _aaa--;
        if (_aaa == 0)
                {
                if (DAC_READY == FALSE)
                        {
                        T1PWMH = _pwmhi; // Write first the high byte to avoid an early PWM
trigger
                        T1PWML = _pwmlo;
                        DAC_READY = TRUE;
                        }
                _aaa = 2;   // 36/2=18khz FSample
                PCOUT = PCOUT ^0x1;             // toggle PC0
```

```
                }
}
```

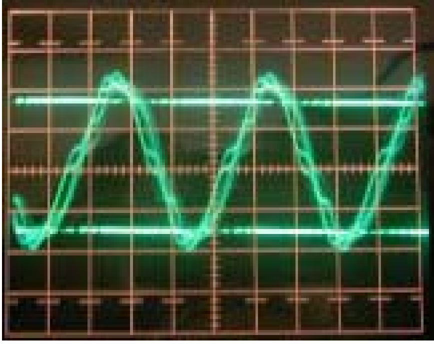
Finally the main loop is implementing the Delta vocoder. The value of the encoded bit can be found at Port C bit 2 for each coding cycle.

```
void main()
{
        adc_initialize();
        dac_initialize();

        invvv = 0x080;
        dac_api(invvv);                                 // Load new input FROM DAC.

        delta = 0;
        while (delta != 32767)
        {
                if ( DAC_READY == TRUE )                // Check for DAC_READY
                {
                        invvv = invvv + 16;
                        if (invvv > 0xff)
                        {
                        invvv = 0x0;
                        }

                delta++;
                }
        dac_api(invvv);                 // Load new input to DAC.
        }


// encoder init
delta = 0;
outd = 0;
histo1 = 1;
histo2 = 0;
syll = 0;


        while (1)   // encoder main loop
        {

        if (invvv > delta)
                        {
                        histo2 = histo1;
                        histo1 = outd;
                        outd = 1;
                        if (outd && histo1 && histo2)
                                syll = syll + int2;
                                else
                                {
                                syll = syll - int2;
                                if (syll<0)
                                        syll=0;
                                }
```

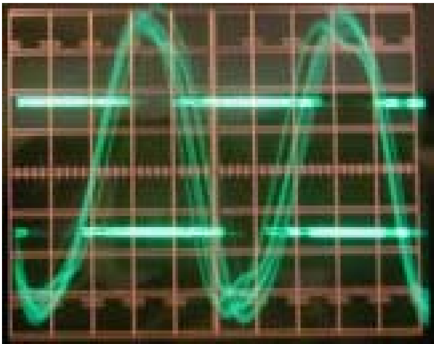
```
                delta = delta + int1 +syll;
                if ( delta > 127)
                        delta=127;
                }
                else
                {
                histo2 = histo1;
                histo1 = outd;
                outd = 0;
                if (outd==0 && histo1==0 && histo2==0)
                        syll = syll+int2;
                        else
                        {
                        syll=syll-int2;
                        if (syll < 0 )
                                syll = 0;
                        }
                delta = delta - int1- syll;
                if (delta < -127)
                        delta = -127;
                }


        PCOUT = (PCOUT & 0xfb)|(outd<<2);       // send outd to port c 2
                while ( DAC_READY != TRUE )             // Wait for DAC_READY
                {;
                }
        dac_api((unsigned int)(delta+128));             // Load new input to DAC. -127 127
>>> 0 255
        invvv = ADCD_H;                         // Use ADC Last conversion, 8bitHigh
        invvv = invvv -128;             // 0-255 to -127 127


        }               // while(1)


}       //main
```

RESULTS

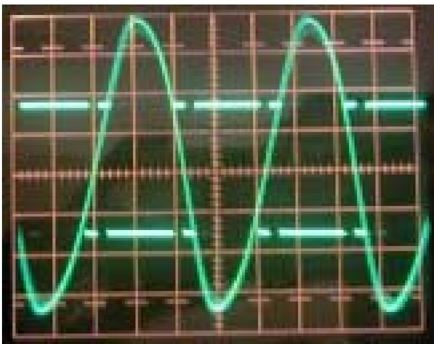
The results were obtained by applying a sine wave to the vocoder input and watching the output on a scope.



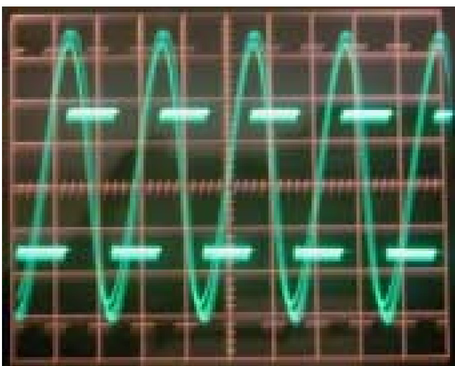
500Hz Sine Wave



1000Hz Sine Wave



1125 Hz Sine Wave



2250 Hz Sine wave

Conclusion

Delta adaptive encoding is a simple method with acceptable results. The implementation using the Z8Encore ADC and PWM is a positive result.

I will attach 3 folders – one is the complete program for the adaptive delta at 18Khz sampling frequency, another is a simple Delta at 18Khz and the third is a simple pass thru 8bit/9Khz. I have used the last one to validate the overall quality of the Anti Alias Filter + ADC + PWM + Anti Alias Filter with 8 bit samples – the result is great. These files can be used as a base to implement other functions like DTMF generator/decoder, etc.

REFERENCES

Harris - Delta Modulation For Voice Transmission / AN607.1
MxComm - Continuously Variable Slope Delta Modulation: A Tutorial
Motorola – Design of Continosly Variable Slope Delta Modulation Communication Systems
Zilog – Z8Encore Documentation and Application Notes

REFERENCES for the Anti Alias Filter:

www.beis.de - Active LPFilter html (Java) Utility
www.catena.uk.com - Simetrix Intro Spice Clone

www.ti.com - Applications Notes:

- sloa088
- sloa049b
- sloa062
- sloa070
- sloa093
- sloa096

www.microchip.com - Application Notes

- Mcp6022
- AN006996
- FilterLab Software

Software Listing

D1800.xls excel simulator listing

```
' Continuously Variable Slope Delta modulation - CVSD
' invvv - input value
' delta - main integrator value
' histo1  - previous output
' histo2 - previous previous output
' outd  - delta output
' int1 - integrator 1 const
' int2 - integrator2 const
' outdold - temp value for outd


Public invvv, delta, histo1, histo2, outd, syll, int1, int2, outdold As Integer



' Continuously Variable Slop Delta modulation - CVSD
Sub main()
Dim a As Integer
int1 = Worksheets("Sheet1").Range("B7").Value       ' first integrator constant
int2 = Worksheets("Sheet1").Range("B8").Value       ' second integrator constant
invvv = 0
delta = 0
outd = 0
histo1 = 1
histo2 = 0
syll = 0

For a = 2 To 300  ' first to last row ENCODE
d1800 (a)
Next a

int1 = Worksheets("Sheet1").Range("B7").Value       ' first integrator constant
int2 = Worksheets("Sheet1").Range("B8").Value       ' second integrator constant
inval = 0
delta = 0
outd = 0
histo1 = 1
histo2 = 0
syll = 0

For a = 2 To 300  ' first to last row DECODE
Id1800 (a)
Next a
```

```
End Sub
Sub d1800(rrr As Integer)
' Delta Adaptive ENCODER

invvv = Worksheets("Sheet1").Range("C" & Trim(Str(rrr))).Value

If (invvv > delta) Then
   histo2 = histo1
   histo1 = outd
   outd = 1

   If (outd And histo1 And histo2) Then
         syll = syll + int2
      Else
         syll = syll - int2
         If syll < 0 Then syll = 0

      End If


   delta = delta + int1 + syll
   If delta > 127 Then delta = 127

   Else    'inval<=delta
   histo2 = histo1
   histo1 = outd
   outd = 0
   If (outd = 0 And histo1 = 0 And histo2 = 0) Then
         syll = syll + int2
      Else
         syll = syll - int2
         If syll < 0 Then syll = 0
      End If

   delta = delta - int1 - syll
   If delta < -127 Then delta = -127
   End If

Worksheets("Sheet1").Range("D" & Trim(Str(rrr))).Value = delta
Worksheets("Sheet1").Range("E" & Trim(Str(rrr))).Value = outd
Worksheets("Sheet1").Range("F" & Trim(Str(rrr))).Value = histo1
Worksheets("Sheet1").Range("G" & Trim(Str(rrr))).Value = histo2
Worksheets("Sheet1").Range("H" & Trim(Str(rrr))).Value = syll
```

```
End Sub


Sub Id1800(rrr As Integer)
' Delta ADAPTIVE DECODER

outdold = outd
outd = Worksheets("Sheet1").Range("e" & Trim(Str(rrr))).Value

If (outd = 1) Then
   histo2 = histo1
   histo1 = outdold


   If (outd And histo1 And histo2) Then
         syll = syll + int2
      Else
         syll = syll - int2
         If syll < 0 Then syll = 0

      End If


   delta = delta + int1 + syll
   If delta > 127 Then delta = 127

   Else    'inval<=delta , outd=0
   histo2 = histo1
   histo1 = outdold

   If (outd = 0 And histo1 = 0 And histo2 = 0) Then
         syll = syll + int2
      Else
         syll = syll - int2
         If syll < 0 Then syll = 0
      End If

   delta = delta - int1 - syll
   If delta < -127 Then delta = -127
   End If

Worksheets("Sheet1").Range("I" & Trim(Str(rrr))).Value = delta
Worksheets("Sheet1").Range("J" & Trim(Str(rrr))).Value = outd
Worksheets("Sheet1").Range("K" & Trim(Str(rrr))).Value = histo1
Worksheets("Sheet1").Range("L" & Trim(Str(rrr))).Value = histo2
Worksheets("Sheet1").Range("M" & Trim(Str(rrr))).Value = syll
```

End Sub

DAC.c – ADC and PWM functions module listing

```
#include <ez8.h>
#include "DAC.h"

unsigned char _pwmlo,_pwmhi, DAC_READY,_aaa ;

void adc_initialize(void)
{

   //PBADDR = 0x01;         // Port B data direction.
   //PBCTL  &= 0xFD;   //
   PBADDR = 0x02;          // Port B alternate function,
   PBCTL  |= 0x01;    // Enabled PB0=ANA0=ADC INPUT 0
          PBADDR = 0x03;            // Port C OPEN DRAIN control
          PBCTL  &= 0xFE;                       // PB0=ANA0 NOT open drain
          PBADDR = 0x04;            // Port C HIGH DRIVE control
          PBCTL  &= 0xFE;                       // PB0=ANA0 NOT High Drive
          PBADDR = 0x0;

          ADCCTL = 0xB0;            // CEN=1, 0, VRED=1, CONT=1, ANAIN=0000
          // CEN will be 0 at first complete conversion result 5129+40 system clock
cycles

          while (ADCCTL & 0x80 )   //wait the end of the first A/D conversion
          {
          ;
          };
}

void dac_initialize(void)
{
          PCADDR = 0x01;            // Port C data direction
          PCCTL  &= 0xF8;           // PC1 = Timer1_out is output. PC0=out,PC2=out,
Rest all inputs.
          PCADDR = 0x02;    // Port C alternate function,
          PCCTL  |= 0x02;    // Enabled Timer1(PWM output)
          PCADDR = 0x03;            // Port C Open Drain control
          PCCTL  &= 0xFD;                       // Timer1(PWM output) NOT open drain
          PCADDR = 0x04;            // Port C HIGH DRIVE control
          PCCTL  |= 0x02;           // Timer1(PWM output) High Drive Enable
          PCADDR = 0x00;            // protect port control
```

```
	/***********************************************************
***
	* Initialize Timer 1 in continuous PWM mode with prescale = 1
	* Initialize the PWM width  0x0100 - 1/2 scale.
	* Initialize the PWM period to 27.77us / 36khz.
	***********************************************************
***/
	T1CTL   = 0x43;     // PWM mode, Prescale 1, TPOL=0, not enabled.
   T1PWMH  = 0x01;     // PWM High
	T1PWML  = 0x00;			// PWM Low        1/2 scale

	T1CPH   = 0x02;			// RELOAD Hi/ Low = 0x0200 with a prescalaer of
1
   T1CPL   = 0x00;			// results 36KHz PWM cycle at 18.432MHz sysclk

	T1CTL  |= 0x80;			// Enable timer PWM

	/***********************************************************
***
	* Initialize Timer 1 Interrupt with high priority.
	* Point interrupt vector to void dac_isr(void);
	***********************************************************
***/
	IRQ0ENH |= 0x40;		// Enable timer 1 interupts as high priority.
	IRQ0ENL |= 0x40;		// INTR is generated as RELOAD value is reached.
	SET_VECTOR (TIMER1,dac_isr);

	DAC_READY = TRUE;
	EI();

	}

void dac_api(unsigned int pinput)
{
	pinput = pinput & 0x0FF;	// Keep only last 8 bits
	pinput = pinput + 0x080;	// Place the 8 bit value 1/2 scale of 9 bit
	_pwmlo = pinput & 0xFF;			// Convert the calculated PWM value
	pinput = pinput >> 8;			// to two bytes and set the PWM hi/low
	_pwmhi = pinput & 0xFF;			// registers.
	DAC_READY = FALSE;

	//T1CTL |= 0x80;		// Enable timer PWM
}
```

```
#pragma interrupt
void interrupt dac_isr(void)
{
		_aaa--;
		if (_aaa == 0)
			{
			if (DAC_READY == FALSE)
				{
				T1PWMH = _pwmhi;		// Write first the high byte to
avoid an early PWM trigger
				T1PWML = _pwmlo;
				DAC_READY = TRUE;
				}
			_aaa = 2; // 36/2=18khz FSample
			PCOUT = PCOUT ^0x1;	// toggle PC0
			}
}
```

MAIN.c - Delta adaptive main module

```
#include <ez8.h>
#include "DAC.h"

// dealta adaptive seech vocoder

extern unsigned char DAC_READY;
int invvv, delta, syll;
unsigned char histo1, histo2, outd, outold;
#define int1 6
#define int2 2

void main()
{
		adc_initialize();
		dac_initialize();

		invvv = 0x080;
		dac_api(invvv);				// Load new input FROM DAC.

		delta = 0;
		while (delta != 32767)
		{
			if ( DAC_READY == TRUE )			// Check for
DAC_READY
			{
```

```
                                invvv = invvv + 16;
                                if (invvv > 0xff)
                                {
                                invvv = 0x0;
                                }

                        delta++;
                        }
                dac_api(invvv);                 // Load new input to DAC.
                }


        // encoder init
        delta = 0;
        outd = 0;
        histo1 = 1;
        histo2 = 0;
        syll = 0;


                while (1) // encoder main loop
                {

                if (invvv > delta)
                                {
                                histo2 = histo1;
                                histo1 = outd;
                                outd = 1;
                                if (outd && histo1 && histo2)
                                        syll = syll + int2;
                                        else
                                        {
                                        syll = syll - int2;
                                        if (syll<0)
                                                syll=0;
                                        }
                                delta = delta + int1 +syll;
                                if ( delta > 127)
                                        delta=127;
                                }
                                else
                                {
                                histo2 = histo1;
                                histo1 = outd;
                                outd = 0;
                                if (outd==0 && histo1==0 && histo2==0)
```

```
                    syll = syll+int2;
                    else
                    {
                    syll=syll-int2;
                    if (syll < 0 )
                              syll = 0;
                    }
          delta = delta - int1- syll;
          if (delta < -127)
                    delta = -127;
          }


PCOUT = (PCOUT & 0xfb)|(outd<<2);          // send outd to port c 2
          while ( DAC_READY != TRUE )                    // Wait for
DAC_READY
          {;
          }
dac_api((unsigned int)(delta+128));          // Load new input to DAC. -127
127 >>> 0 255
invvv = ADCD_H;                              // Use ADC Last conversion,
8bitHigh
invvv = invvv -128;                // 0-255 to -127 127


}          // while(1)


}          //main
```