MOTOROLA
Microprocessor and Memory
Technologies Group

AN328-PW

Preliminary

Application Note

Generating DTMF with PWM module of Dragon-

INTRODUCTION

Telephones and associated ancillary equipment providing intelligent features are fast becoming commonplace. Frequently, it is necessary for the microprocessor to provide the intelligence to also dial a telephone number.

The DragonBall™ MC68328 microprocessor, with its proven hardware and software versatility, is an ideal candidate for such applications. Illustrated here is a cost-effective method of telephone dialling. There are two methods to dial a telephone number, the Rotary Pulse Dialling and the Dual Tone Multi-Frequency (DTMF) Tone Dialling. Although pulse dialling is much more simple to implement from both hardware and software viewpoints, DTMF tone dialling is introduced here. Tone dialling has several advantages over pulse dialling:

- Tone dialling requires a much shorter dialling time.
- Pulse dialling requires a direct connection to the telephone line while tone dialling does not.
- Tone dialling is more common in telephone systems nowadays, especially in automatic (machine) answering systems.

DTMF DIALLING

DTMF tone dialling requires the MC68328 to generate two simultaneous sine waves of different frequencies. Table 1 below shows the keypad digits and frequencies of the corresponding tone pairs. They fall into two major groups:

GROUP	*FREQUENCY(Hz)*
Low Tones	697,770,852,941
High Tones	1209,1335,1477,1633

Keypad Digit	Tone Pair (Hz)
0	941,1335
1	697,1209
2	697,1335
3	697,1477
4	770,1209
5	770,1335
6	770,1477
7	852,1209
8	852,1335
9	852,1477
A	697,1633
B	770,1633
C	852,1633
D	941,1633
*	941,1209
#	941,1477

Table 1. Keypad Digits and Frequencies of Tone Pairs

The frequencies and the keypad layout of DTMF tone dialling has been internationally standardized, but the tolerances on individual frequencies may vary in different countries. The North American standard is 1.5% for the generator and 2% for the receiver. Since the DTMF tone is generated by the Pulse Width Modulation (PWM) module and the timer interrupt, the accuracy of the frequencies generated is directly affected by the accuracy of the frequency of the crystal oscillator of the system. If the system uses a different clock rate, the source code has to be adjusted accordingly.

Moreover, the tone has to be held for a specific minimum time before it is accepted as a valid dialling digit. In North America, the minimum time for a digit is about 50 ms and the inter-digit interval is also about 50 ms.

GENERATING DTMF BY PWM

The DTMF tone is generated by the Pulse Width Modulation (PWM) module of the Dragonball™ MC68328 microprocessor. The software will modulate the sinusoidal signal into a pulse train of fixed periods but changing width. The changing width of the pulses corresponds to the voltage level of the sine wave. With an external Low Pass Filter (LPF) at the PWMOUT pin, the PWM signal will be demodulated. The LPF acts as an integrator which transforms the pulse train into analog sinusoidal signal. The DTMF waveform will then sent to the audio amplifier for sound output. The flowchart of the process is shown in Figure 1.

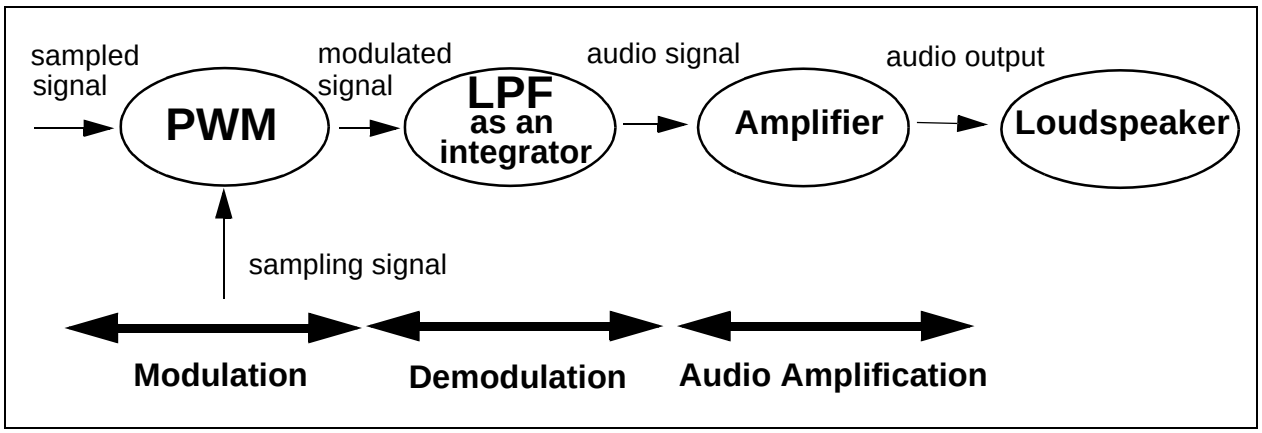

Figure 1. Block Diagram of DTMF Generation

The modulation process is done by software. As explained previously, DTMF tone dialling requires the generation of two sine waves. The basic concept of modulation is to use two counters to increment through a sine wave table at different speed. To generate lower frequencies one steps through the table in smaller steps. Similarly, higher frequencies require larger steps. The two sine waves are superimposed to give the desired DTMF waveform. The process is depicted in Figure 2.

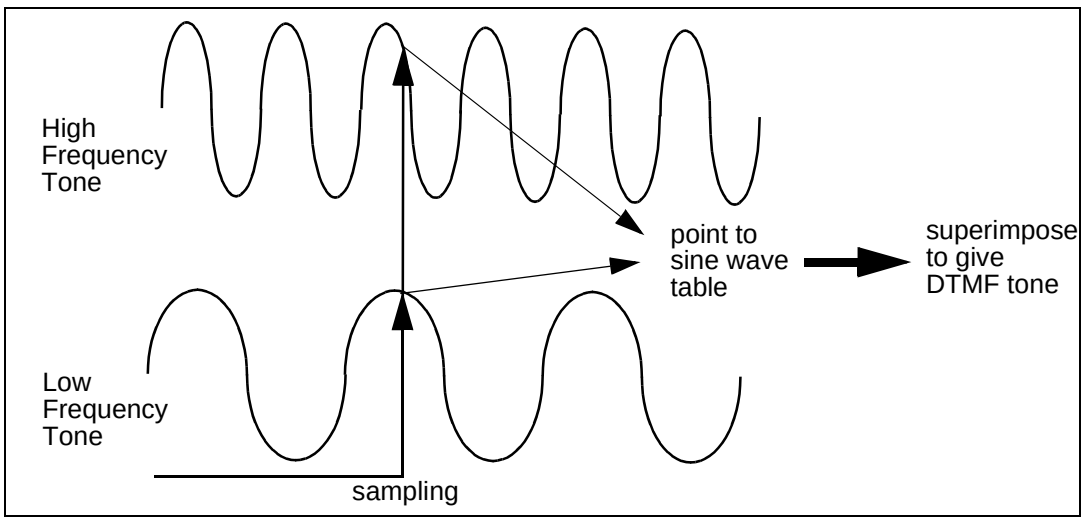

Figure 2. Modulation Process of PWM

As shown in Figure 3, a 128-entry table is created to represent a single period of a complete sine wave.

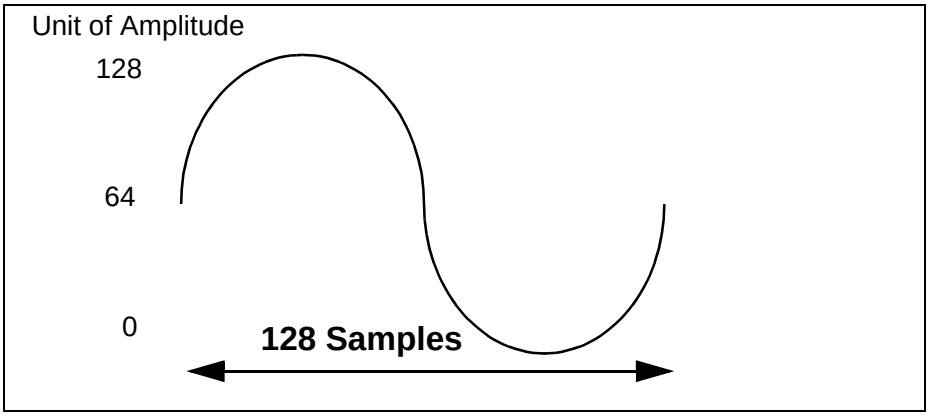

Figure 3. Sine Wave Table Representation

The table can be generated in C format as follows:

```
main()
{
    unsigned float pi = 3.141596;
    unsigned int i;

    printf("SINE_TABLE\n");
    printf("---------------------");
        for (i = 0 ; I < 128 ; i++ )
          {
          printf("%3d, ",(int)(sin(2*i*pi/128) * 64 + 63)); i++;
          printf("%3d, ",(int)(sin(2*i*pi/128) * 64 + 63)); i++;
          printf("%3d, ",(int)(sin(2*i*pi/128) * 64 + 63)); i++;
          printf("%3d, \n",(int)(sin(2*i*pi/128) * 64 + 63));
          }
}
```

Figure 4. Generation of Sine Wave Table using C

The 128-entry table generated by the C code is shown in Table 2.

63	66	69	72
75	78	81	84
87	90	93	95
98	101	103	105
108	110	112	114
116	117	119	120
122	123	124	125
125	126	126	126
126	126	126	126
125	125	124	123
122	120	119	117
116	114	112	110
108	105	103	101
98	95	93	90
87	84	81	78
75	72	69	66
63	59	56	53
50	47	44	41
38	35	32	30
27	24	22	20
17	15	13	11
9	8	6	5
3	2	1	0
0	0	0	0
0	0	0	0
0	0	1	2
3	5	6	8
9	11	13	15
17	20	22	24
27	30	32	35
38	41	44	47
50	53	56	59

Table 2. 128-entry Sine Wave Table

It is worth noting that a table of more than 128 entries may also be used to enhance the accuracy of the output. For instance, a 256-entry or even a 65536-entry table can be made. The source code should then be adjusted accordingly to adapt to the new format of the table.

CALCULATION OF INCREMENT VALUE

The PWM period register is set to have a fixed value of 256 counts with a prescaler value of 4. It means that the PWMOUT pin will send a pulse in every 61.50 μs (which corresponds to 1024 clock cycles) under a 33.3 MHz oscillator frequency . Meanwhile, the timer interrupt is set to have an interrupt every $400 (=1024) clock cycles and the Interrupt Service Routine (ISR) put the calculated pulse width into the PWM width register. PWM module will then automatically send the pulse with width corresponding to the value of that register .

As explained previously, the calculation of the pulse width requires incrementing the two counters. This process is depicted in Figure 5.

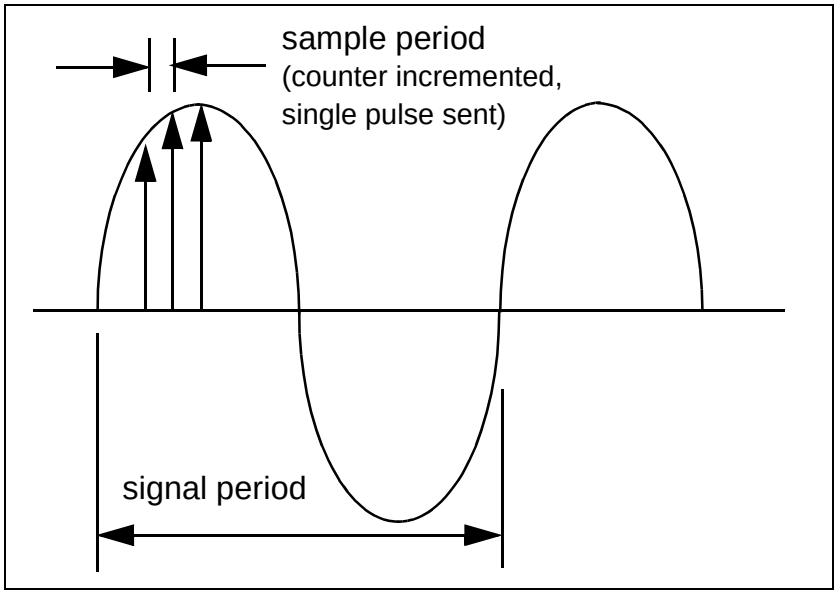

Figure 5. Calculation of Increment Value

The increment value is calculated as follows.

We assume that a complete sine wave has 65536 (2^{16}) intervals, so

increment value = 65536 / number of increments

To calculate the number of increments, we consider the sampling period and the signal period.

number of increments = how many times the given signal increments through the sine wave table in one complete cycle

= signal period / sample period

In other words,

number of increments = sample frequency / signal frequency

Thus, we have

increment value = 65536 * signal frequency / sample frequency

Example: Calculation of Increment Value

Suppose we want to generate a 697 Hz audio signal.

The oscillator frequency is 33.3 MHz.

Procedure 1:	Signal frequency	= 697 Hz
Procedure 2:	Sample frequency	= (33.3 MHz / 2) / 1024 clock cycles per interrupt
		= 16259.8 Hz
Procedure 3:	Applying the formula,	
	Increment Value	= 65536 * 697 / 16259.8
		= 2809.3

This value will be used to increment the counter for looking up values from the sine wave table. However, as we use a 128-entry sine wave table instead of a 65536-entry one, this value should be scaled by taking the most significant 7 bits only. The Increment Value for all the frequencies of the DTMF are shown below in Table 3.

Frequency (Hz)	Increment
697	2809.3
770	3103.5
852	3434.0
941	3792.8
1209	4872.9
1335	5380.8
1477	5953.1
1633	6581.9

Table 3. Calculation of Increment Value

DEMODULATION OF PWM

Demodulation of the PWM output back into DTMF tone requires the use of a Low Pass Filter (LPF). The LPF works as an integrator which integrates the energy of the pulse train into analog sinusoidal signal. Since the pulses consist of square waves, which have significantly large components of high frequency signal, it requires a higher order LPF to accomplish the task of demodulation. To avoid the use of inductors in the circuit, active LPF using LM324 operational amplifiers is chosen. Two second-order active LPF's are cascaded to provide a fourth-order filtering. The circuit of the LPF is shown below in Figure 6.

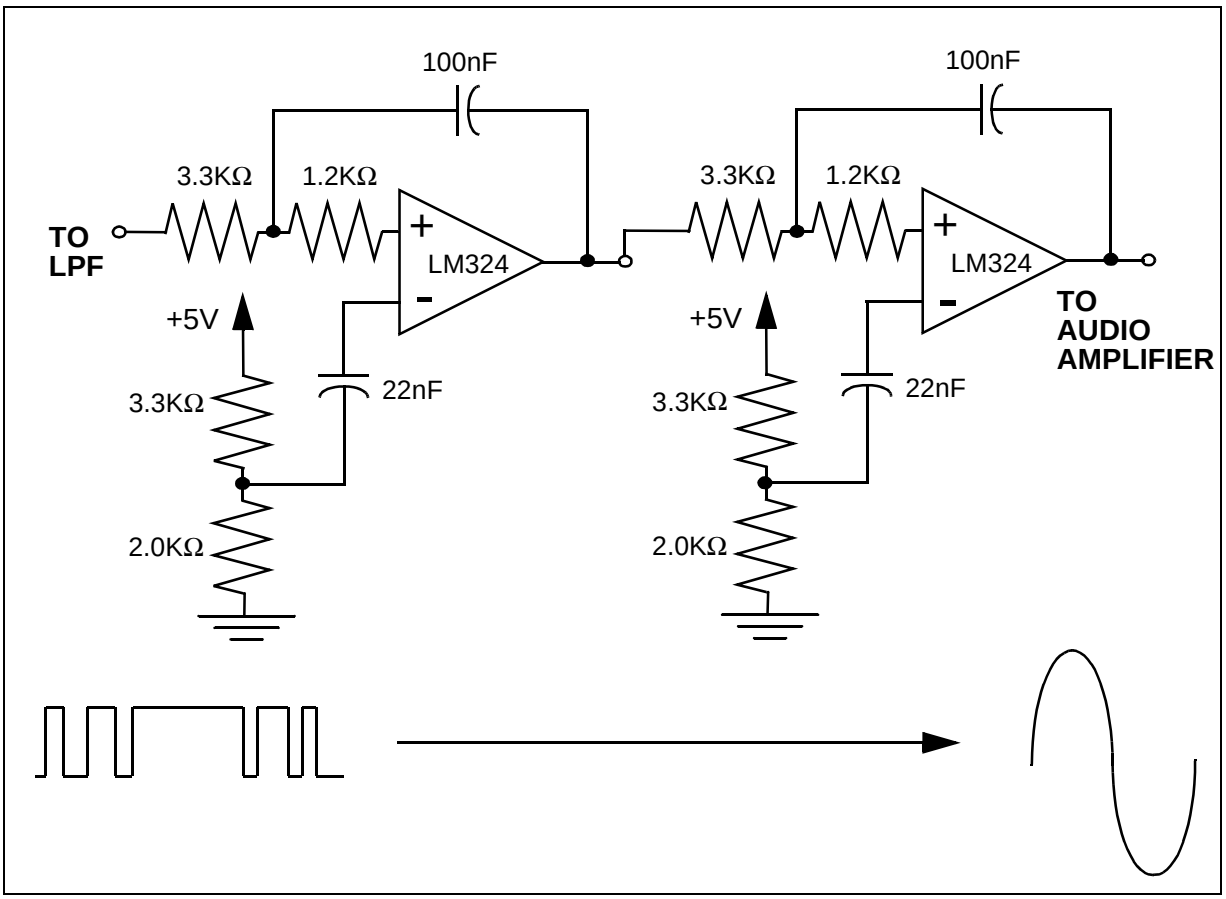

Figure 6. Fourth-order Active Low Pass Filter

Since the DTMF tone ranges from 697 Hz to 1633 Hz, the cutoff frequency of each active LPF is chosen to be at about 1.7 KHz, and the quality factor is about unity. Moreover, since the operational amplifiers are powered by Vcc of the system (+5V) and the gain in the pass band of the filter is unity, the operational amplifier is operating in a range very close to saturation. To avoid distortion in the signal, a potential divider followed by a voltage follower is added inbetween the PWMOUT pin and the LPF. The circuit diagram of the potential divider is shown in Figure 7.

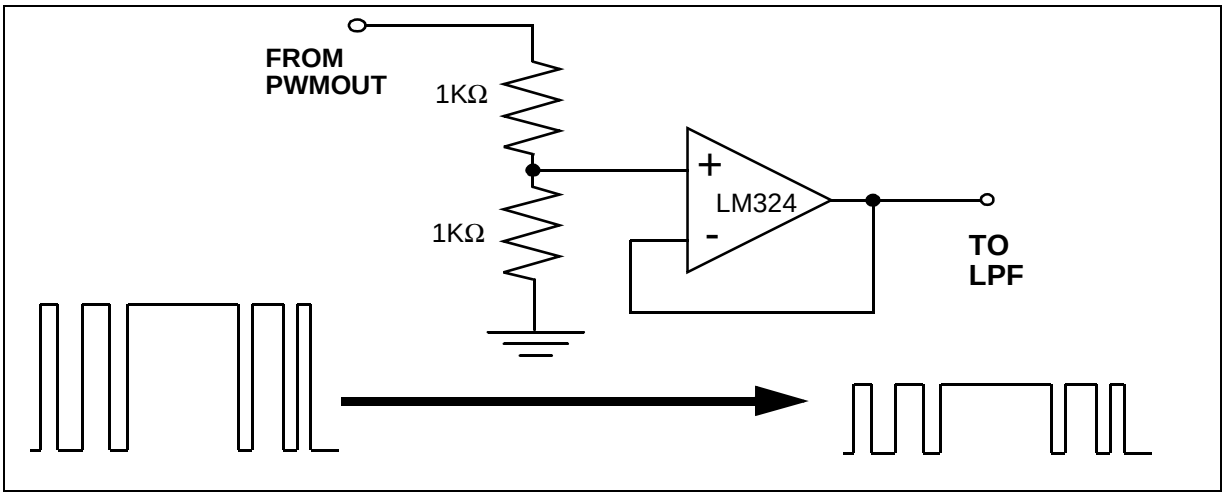

Figure 7. Potential Divider plus Voltage Follower

A single LM324 operational amplifier chip has four independent operational amplifiers. Hence, the voltage follower and the two active LPF's can share the same LM324 chip.

The DTMF waveform generated is shown below in Figure 8 and 9. In Figure 8, we can measure the signal-to noise-ratio by reading the frequency spectrum. The signal-to-noise ratio is approximately 34 dB. In Figure 9, we can see that there are two impulse in the frequency spectrum. The two frequencies, 852 Hz and 1477 Hz, are the dual tone for the digit 9. All the graphs of waveforms presented in this application note are generated by Tektronix TDS 644A Digitizing Oscilloscope.

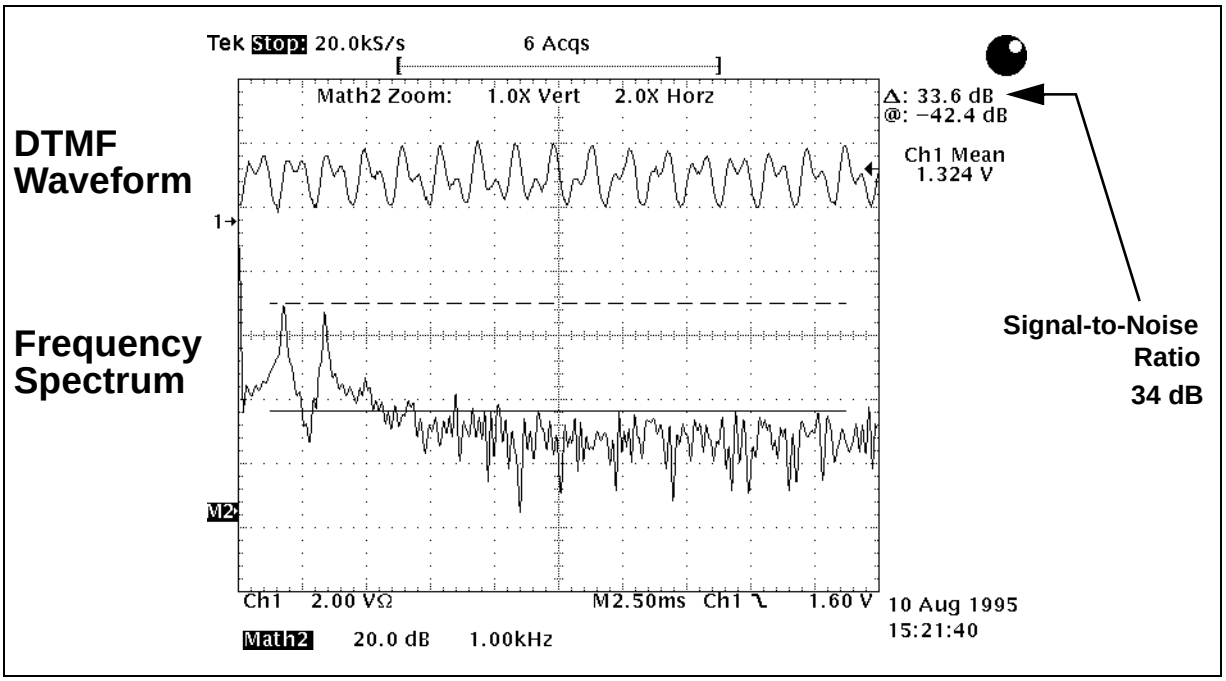

Figure 8. DTMF Waveform and its Frequency Spectrum

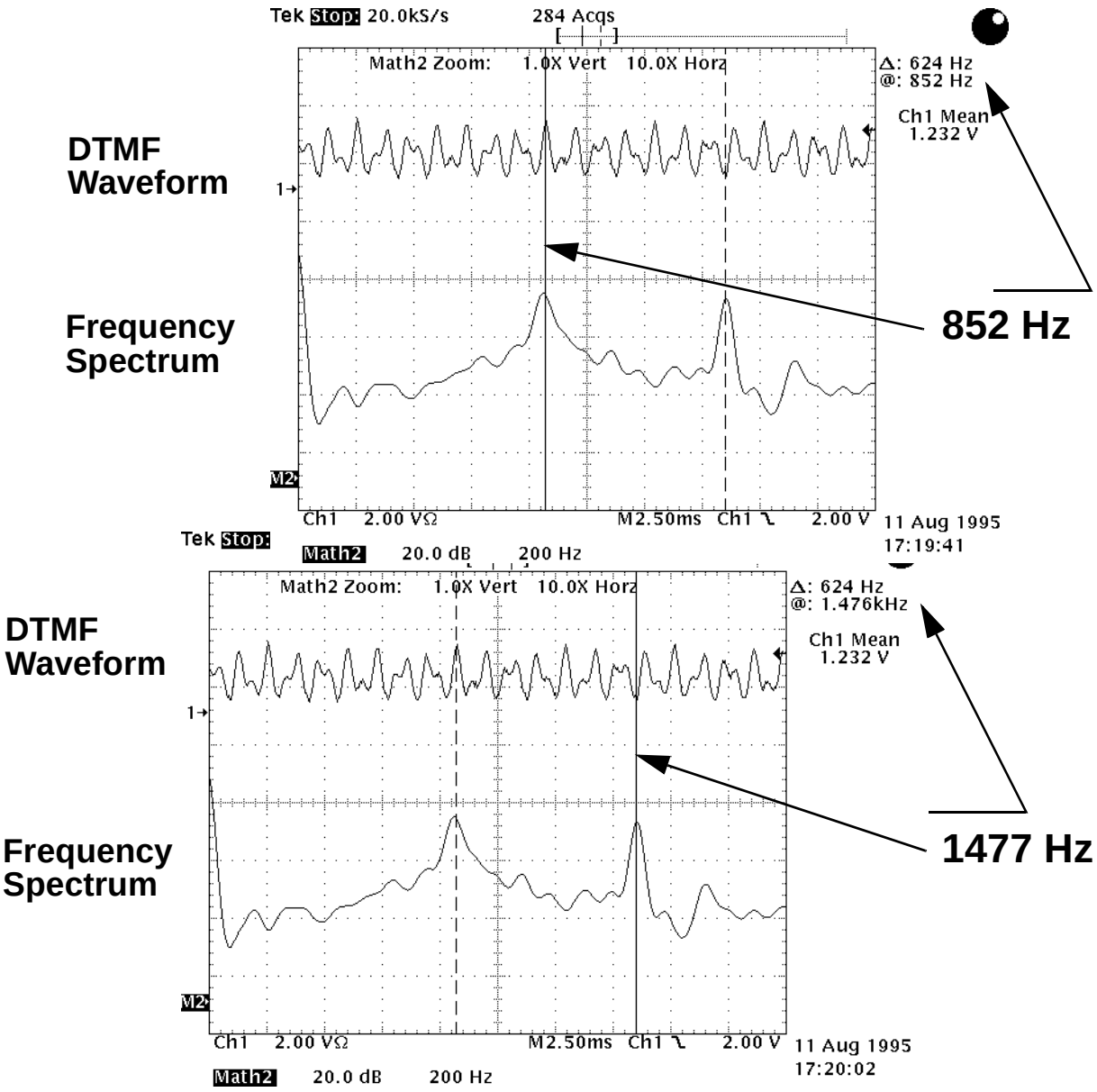

Figure 9. DTMF Waveform and its Frequency Spectrum

AUDIO AMPLIFICATION

The output of the LPF has a moderate impedance and is not very suitable for driving the loudspeaker directly. Hence, an audio amplifier is added.

The MC34119 Low Power Audio Amplifier is chosen for this application. This amplifier has the Chip Disable (CD) pin, which can be set high to shut down the amplifier when it is not in use, thus minimizing the power consumption of the system. The circuitry connecting to the chip is shown below in Figure 10.

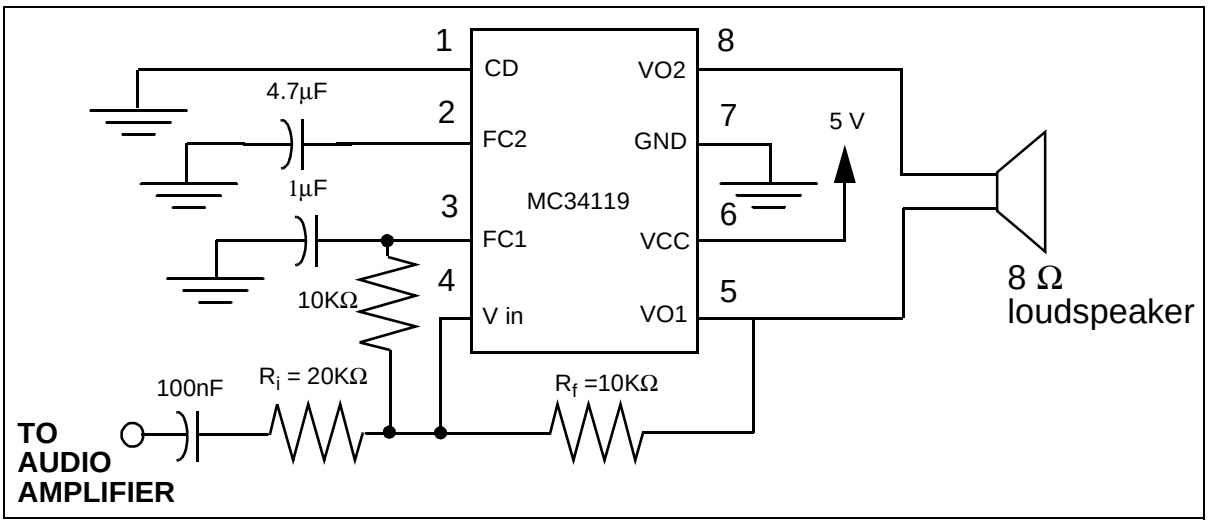

Figure 10. MC34119 Low Power Audio Amplifier

The differential gain of the audio amplifier is calculated by

$$DifferentialGain = 2 \times \frac{R_f}{R_i}$$

which is unity in this example. A variable resistor can be used to control the volume.

It is worth mentioning that a resistor is inserted across pin 3 and pin 4. It is applied to allow the capacitor to charge up or discharge so as to eliminate the "pop" sound when enabling or disabling the audio amplifier. For more details, please refer to Motorola's Application Note AN1081 - 'Minimizing the "pop" in the MC34119 Low Pwer Audio Amplifier'.

A very important issue has to be mentioned here. For good performance, the designer must ensure that the loudspeaker is mounted securely. Insecure mounting will cause the loudspeaker to recoil when the tone is generated. This will severely affect the quality of the sound produced. DTMF tone of poor quality will be rejected by the central telephone office as an invalid tone.

There is more issue on the audio amplification. Care should be taken when designing the power source to the audio amplifier. The audio amplifier, together with the loudspeaker, is a heavy current consumer. When the audio amplification circuit is enabled, the total power consumption of the system will increase significantly. This will endanger the stability of the voltage output of the power source. Therefore, the audio amplifier circuit should be suitably bypassed with capacitors.

ALTERNATIVE SOLUTION FOR DEMODULATION AND AUDIO AMPLIFICATION

Since the MC34119 Low Power Audio Amplifier may be configured to provide the feature of bandpass filtering, we can use a simplier LPF together with the audio amplifier in bandpass configuration. This will eliminate the necessity of the active filters, thus saving us an LM324 operational amplifier module and many resistors and capacitors.

A simpler first-order LPF is chosen here. The circuit diagram is shown in Figure 11.

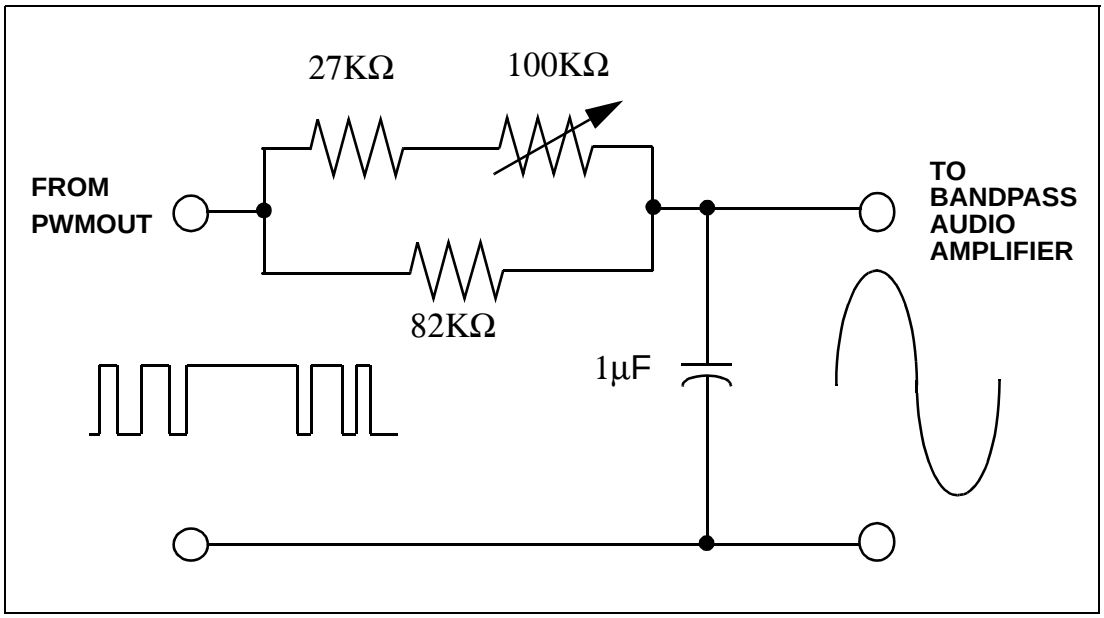

Figure 11. Simple First-order Low Pass Filter

The cutoff frequency is chosen to be lower than that of the DTMF tone.

$$\tau = RC = 20 \text{ ms} - 50 \text{ ms}$$

$$f = 50 \text{ Hz} - 20 \text{ Hz}$$

In such a configuration, the DTMF tone falls into the downward sloping region of the filter's Bode plot, which acts more like an integrator. A variable resistor is added here for volume control. It is because the differential gain in the audio amplifier in bandpass configuration has to be adjusted by changing the value of more than one components. This makes volume control very inconvenient. On the contrary, changing the value of the resistor in the LPF can adjust the volume without significantlyaffecting the LPF's behaviour as an integrator. The audio amplifier is chosen to have bandpass filtering with the pass band from 20 Hz to 20 KHz, which is the audible range of the human ear. The circuit connecting to the MC34119 Low Power Audio Amplifier is shown in Figure 12.

.

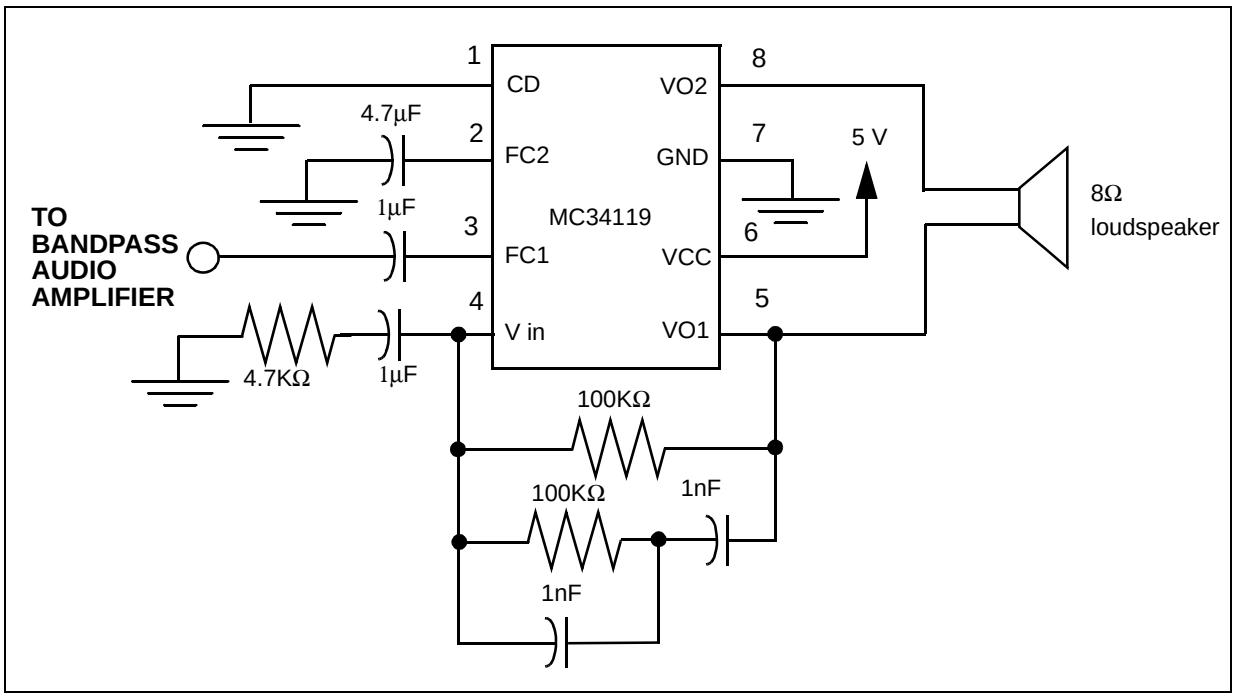

Figure 12. MC34119 Audio Amplifier in Banpass Configuration

The DTMF waveform generated at the audio amplifier output is shown in Figure 13. Using this method, the signal-to-noise ratio is approximately 29 dB.

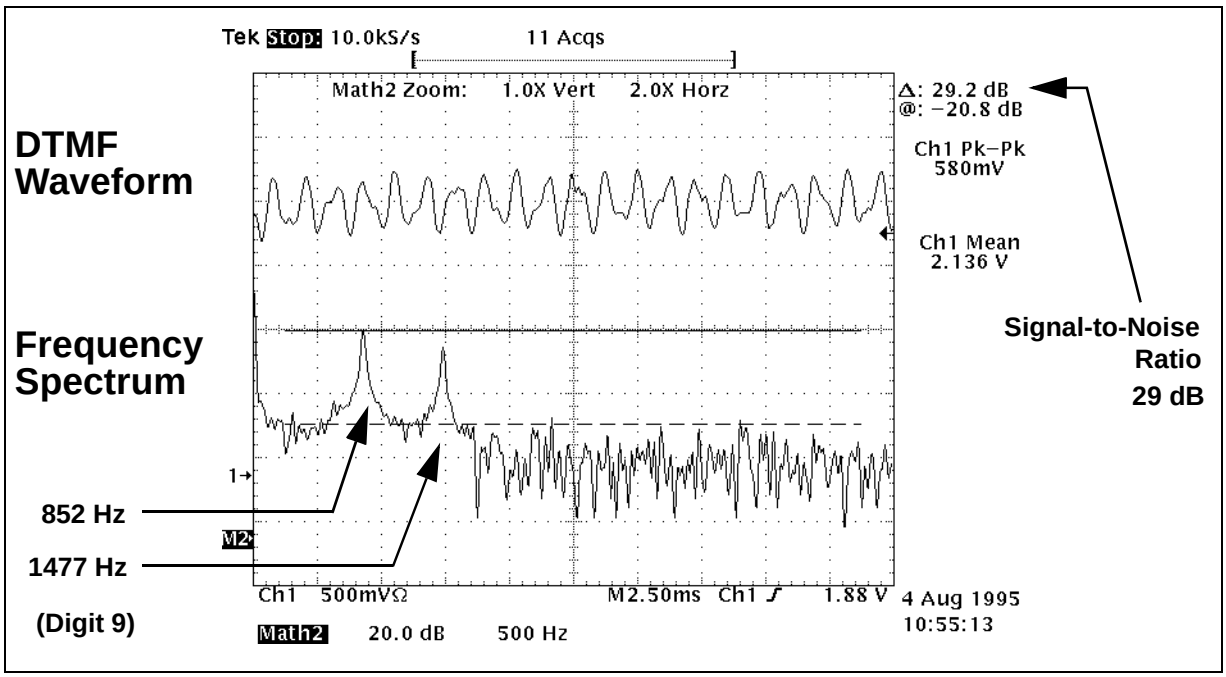

Figure 13. DTMF Waveform and its Frequency Spectrum

WARM-UP PROBLEM

In many audio signal generation, the waveform has an average value at ground level, which is 0V. The voltage varies between positive and negative values. However, in our application using PWM, the voltage level can only be positive. It means that the waveform is generated with a DC offset. The situation is depicted in Figure 13.

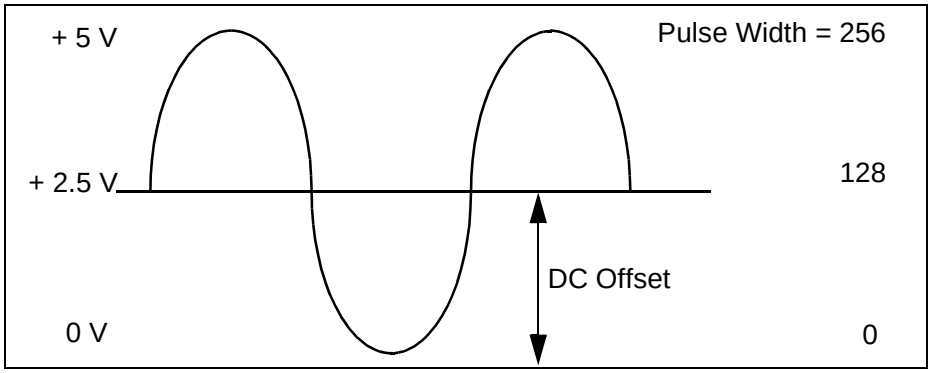

Figure 14. PWM Voltage level

Before the module is enabled, the PWMOUT sends no pulse (zero pulse width). The demodulated signal remains at ground and each pin of the differential output of the audio amplifier is 2.5 V. However, when the PWM is enabled, instead of zero pulse width, the pulse width of 128 will represent the 2.5 V audio output level. Thus, there is a jump of the reference pulse width. If we increase the reference pulse width instantaneously, there will be a sudden rise in the audio output. In other words, the differential output of the audio amplifier will have many sharp impulses at the moment when the PWM module is enabled or disabled. The signal is shown in Figure 15.This effect is undesirable because it produces very low frequency “hum” sound in the loudspeaker, and the impulses drain very large current, endangering the stability of the power supply.

To tackle this problem, software code is written to warm-up the audio output so that the reference pulse width can be gradually increased from 0 to 128. When the PWM module is enabled, it is asked to send pulses of linearly increasing width from 0 to 128. As shown in Figure 16, there is a trade-off between the warm-up time and the smoothness of the waveform. The longer the warm-up time, the smoother the waveform, and vice versa.In-between any two digits, instead of disabling and enabling the PWM module, we send pulses of constant width of 128 so as to keep the voltage level stably at 2.5V.

Although the North American standard is 50 ms for each digit and another 50 ms for each inter-digit interval, experiment shows that the minimum time required for each digit and that for the inter-digit interval are both approximately 30 ms.

In this program, the warm-up time we have chosen is approximately 1 second. Also, each digit is chosen to last for 90 ms, and the pulse between two digits 50 ms. The complete signal is shown in Figure 17. The total time required to dial a 9-digit telephone number is

*Time required = warm-up time + 9 digits * tone time + 8 pauses * pause time*

*= 1 sec + 9 * 90 ms + 8 * 50 ms*

= 2.2 secs

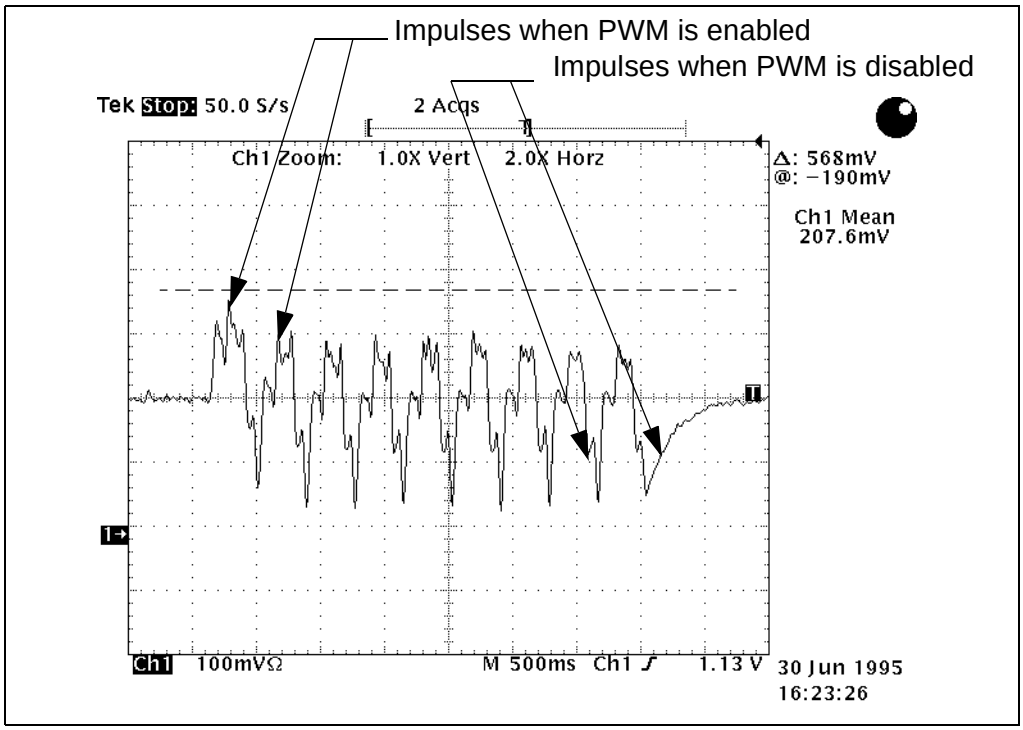

Figure 15. Voltage Impulses at Audio Output when the PWM is enabled or disabled

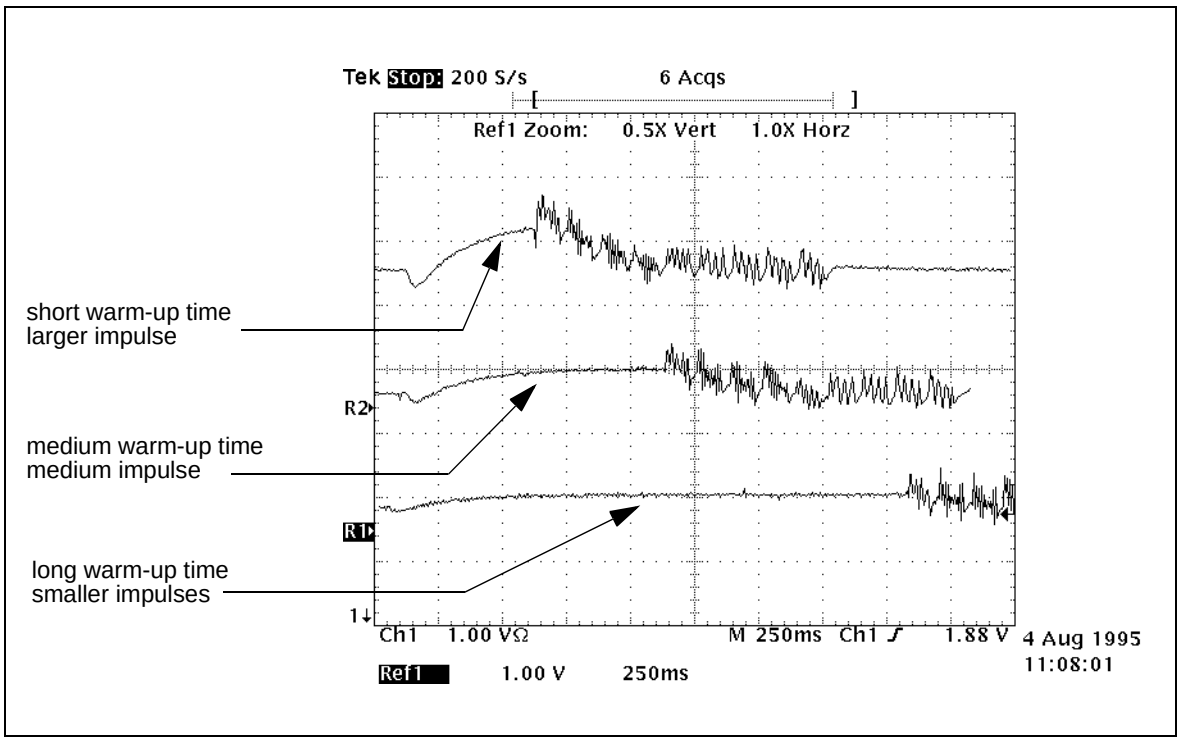

Figure 16. Relationship between Warm-up Time and Smoothness of the Output Signal

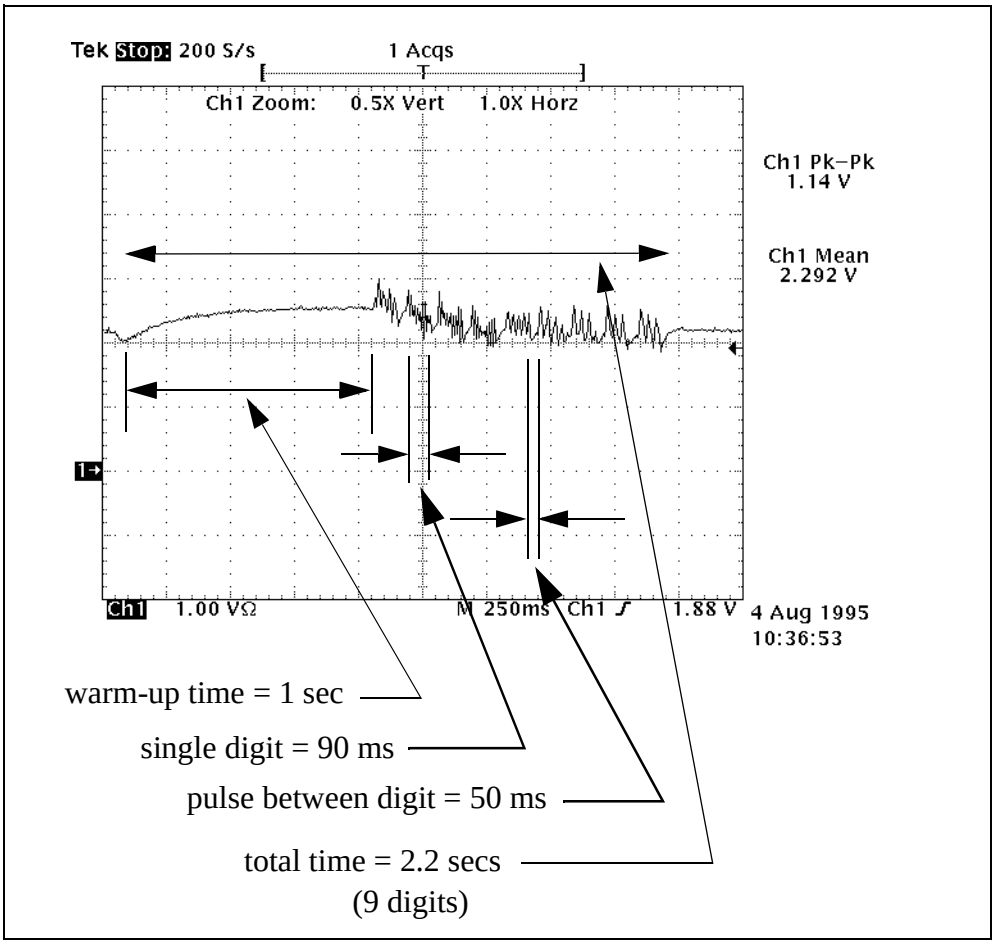

Figure 17. Time Required to Dial a Telephone Number

CONCLUSION

In this aplplication note, we have investigated the hardware and software design criteria on using the PWM module of the DragonBall™ MC68328 microprocessor to generate the DTMF tone. Using the same method, theoretically we can generate any signal of bandwidth smaller than the sampling frequency of the PWM module. With minor adjustment, we may use the PWM module to synthesize human speech.

APPENDIX: SOURCE CODE

The source code for the DTMF tone dialling is provided here. The phone number to be dialled is stored in the variable PHONE_NO, which is set to 926668333 in this example. Telephone number of any length may be dialled. Digit 0 - 9 are represented in hexadecimal as $00 - $09, while $0A - $0F represents the digit A, B, C, D, * and # respectively. The phone number should be terminated by the byte $10.

```
* Program Description:
*
* This program generates the DTMF tone for telephone dialling
* It uses the PWM module of the DragonBall  MC68328

**********************
*     Definitions     *
**********************

M328_BASE       EQU     $FFFFF000       ;base address of PWM

* PWM address
PWMC            EQU     M328_BASE+$500  ;PWM control register
PWMP            EQU     M328_BASE+$502  ;PWM period register
PWMW            EQU     M328_BASE+$504  ;PWM width register
PWMCNT          EQU     M328_BASE+$506  ;PWM counter register

* Timer
TCTL1           EQU     M328_BASE+$600  ;Timer Unit 1 Control Register
TPRER1          EQU     M328_BASE+$602  ;Timer Unit 1 Prescaler Register
TCMP1           EQU     M328_BASE+$604  ;Timer Unit 1 Compare Register
TCR1            EQU     M328_BASE+$606  ;Timer Unit 1 Capture Register
TCN1            EQU     M328_BASE+$608  ;Timer Unit 1 Counter Register
TSTAT1          EQU     M328_BASE+$60A  ;Timer Unit 1 Status Register

* Interrupt request
IVR             EQU     M328_BASE+$300  ;Interrupt Vector Register
ICR             EQU     M328_BASE+$302  ;Interrupt Control Register
IMR             EQU     M328_BASE+$304  ;Interrupt Mask Register
IWR             EQU     M328_BASE+$308  ;Interrupt Wakeup Enable Register
ISR             EQU     M328_BASE+$30C  ;Interrupt Status Register
IPR             EQU     M328_BASE+$310  ;Interrupt Pending Register

DISABLEINTS     EQU     $2700           ;INTERRUPT MASK = 7
ENABLEINTS      EQU     $2500           ;INTERRUPT MASK = 5

DISABLEPWM      EQU     $0000           ;PWMmask,PWMEN=0
ENABLEPWM       EQU     $0010           ;PWMmask,PWMEN=1

* ************************************************************************** *

                SECTION code

*************************************
*     Programme Initialization      *
*************************************

START:
* Disable Interrupts
           move.w #DISABLEINTS,sr       ;disable interrupts
```

```
* Initialize PWM
          move.w #256,PWMP            ;set period register to 1024 clock cycles

* Initialize Timer
          move.w #$0033,TCTL1         ;disable interrupt on capture event
                                      ;active-low pulse for one SYSCLK period
                                      ;enable interrupt on reference event
                                      ;restart mode, timer Enable
                                      ;33.3MHz clock
          move.w #$0000,TPRER1        ;prescaler divide value to 1
          move.w #$0400,TCMP1         ;set compare register to 1024 clock cycles

* Initialize Interrupt Vector
          ori.b #$a8,IVR
          andi.b #$af,IVR             ;set IVR = 10101 110(level 6)
                                      ;ICR is ignored
          andi.l #$ffbfffff,IMR       ;enable timer 1 interrupt
          ori.l #$00400000,IWR        ;enable timer 1 wakeup sequence

* Reset Interrupt Level
          andi.w #$f5ff,sr            ;set interrupt level to 5

* Define Autovector *
* IVR = 10101 110  (interrupt vector )
* multiply by 4 to get the address: 10 1011 1000 = $2b8
          movea.l #$2b8,a1            ;the calculated vector is $2b8
          move.l #GEN_DTMF,(a1)       ;set autovector

* Initialize Variables
          move.w #0,FREQ1             ;clear FREQ1
          move.w #0 ,FREQ2            ;clear FREQ2
          move.l #0,COUNT             ;clear COUNT
          move.l #0,DIGIT             ;clear DIGIT
          move.w #0,BEGIN             ;clear BEGIN

* Enable Interrupts
          move.w #ENABLEINTS,sr       ;enable interrupts
* Enable PWM
          move.w #0,PWMW              ;clear PWM width
          move.w #ENABLEPWM,PWMC      ;enable PWM

****************************************
*      Warmup for the Audio Amplifier     *
****************************************

* Increase the voltage level linearly from 0V to 2.5V
* Loop for 16284 interrupts  so that PWMW increases from 0 to 128 gradually
* Time required approximately 1 second
loop:
          move.l COUNT,d0
          cmp.l #$4000,d0
          bls loop

          move.w #DISABLEINTS,sr      ;keep PWMW at 128
          move.w #1,BEGIN             ;set flag BEGIN to 1

****************************************
*      Dialling of the Phone Number     *
****************************************
```

```
* Phone number determination
phone:
           move.l #PHONE_NO,a0          ;point to the PHONE_NO sequence
           move.l #TONE_TABLE,a1        ;point to the TONE_TABLE
           add.l DIGIT,a0               ;get the current digit to dial
           move.b (a0),d1
           cmp.w #16,d1                 ;if the digit is 16, end of dialling
           beq loop3

           asl.l #2,d1                  ;multiply the digit by 4 for addressing
           move.l d1,a2
           add.l a1,a2                  ;point to the tone at the TONE_TABLE
           move.w (a2),INC1             ;put the dual-tones into INC1 and INC2
           move.w (2,a2),INC2

           move.w #ENABLEINTS,sr        ;enable interrupts

* Hold the dual-tone of a digit for 0.2 seconds
           move.l #0,COUNT
loop1:
           move.l COUNT,d0
           cmp.l #$600,d0
           bls loop1

           move.w #DISABLEINTS,sr       ;disable interrupts
           move.w #128,PWMW             ;keep pulse width at 128

* No tone for 0.2 seconds between different digits
           move.l #0,d0
loop2:
           add.l #1,d0
           cmp.l #$4000,d0
           bls loop2

* Increment the DIGIT
           add.l #1,DIGIT               ;add DIGIT by 1

* Go to the next digit
           bra phone                    ;go to the next digit

* End of DTMF tone generation
* An infinite loop is placed here to wait for hardware reset
* Should be replaced by the code that returns to the part of the programme that calls the PWM
loop3:
           bra loop3

* ************************************************************************** *

*********************************
*    Interrupt Service Routine   *
*********************************

* If begin flag = 0, programme is in warmup state
* If begin flag = 1, programme is in dialling state

GEN_DTMF:
           move.w #DISABLEINTS,sr       ;disable interrupts
           move.w TSTAT1,d6             ;read the timer status register
           move.w #$00,TSTAT1           ;clear the timer interrupt flag
```

```
            movem.l a0-a3,-(sp)          ;save the address registers
            movem.l d0/d1,-(sp)          ;save the data registers

            add.l #$01,COUNT             ;increment COUNT by 1
            move.w BEGIN,d0              ;check the state of the programme
            cmp.w #0,d0
            bne GEN_DTMF1                ;branch to GEN_DTMF1 for dialling state
            move.l COUNT,d0              ;use COUNT to linearly increase PWMW
            asr.l #7,d0                  ;divide COUNT by 128
            move.w d0,PWMW               ;put into PWMW
            bra END_DTMF                 ;end of ISR

GEN_DTMF1:
            move.l #SIN_TABLE,a0         ;point to the sin table
            move.l #FREQ1,a1             ;point to the first tone counter
            move.l #FREQ2,a2             ;point to the second tone counter
            move.l #PWMW,a3              ;point to the PWMW

            move.w (a1),d0               ;get the old value of tone 1
            add.w INC1,d0                ;increment the tone 1 counter
            move.w d0,(a1)               ;store the new value to tone 1 counter
            and.w #$ff00,d0              ;mask the most significant bits for lookup
            lsr.w #8,d0                  ;obtain the index for lookup in sin table
            lsr.w #1,d0
            move.b (a0,d0),d1            ;get the next sample for tone 1 and save it

            move.w (a2),d0               ;get the old value of tone2
            add.w INC2,d0                ;increment the tone 2 counter
            move.w d0,(a2)               ;store the new value to tone 2 counter
            and.w #$ff00,d0              ;mask the most significant bits for lookup
            lsr.w #8,d0                  ;obtain the index for lookup in sin table
            lsr.w #1,d0
            add.b (a0,d0),d1             ;add the next sample for tone 2 to that of tone 1

            move.w d1,(a3)               ;send the sample to PWMW

END_DTMF:
            movem.l (a7)+,d0/d1          ;restore the data registers
            movem.l (a7)+,a0-a3          ;restore the address registers

            move.w #ENABLEINTS,sr        ;enable interrupts
            rte                          ;end of ISR

* ********************************************************************* *

**********************************
*      Constants and Variables     *
**********************************

FREQ1:      ds.w 1                       ;current location of the first tone
FREQ2:      ds.w 1                       ;current location of the second tone

INC1:       ds.w 1                       ;increment value for the first tone
INC2:       ds.w 1                       ;increment value for the second tone

COUNT:      ds.l 1                       ;counter for loops

BEGIN:      ds.w 1                       ;flag (0 = warmup ; 1 = dialling)
```

```
PHONE_NO:
          dc.b 9,2,6,6,6,8,3,3,3,16    ;phone number to dial 926668333
                                       ;always ends with 16

DIGIT:    ds.l 1                       ;number of digits dialled

SIN_TABLE:                             ;128-entry complete sine wave table
          dc.b 63,66,69,72
          dc.b 75,78,81,84
          dc.b 87,90,93,95
          dc.b 98,101,103,105
          dc.b 108,110,112,114
          dc.b 116,117,119,120
          dc.b 122,123,124,125
          dc.b 125,126,126,126
          dc.b 126,126,126,126
          dc.b 125,125,124,123
          dc.b 122,120,119,117
          dc.b 116,114,112,110
          dc.b 108,105,103,101
          dc.b 98,95,93,90
          dc.b 87,84,81,78
          dc.b 75,72,69,66
          dc.b 63,59,56,53
          dc.b 50,47,44,41
          dc.b 38,35,32,30
          dc.b 27,24,22,20
          dc.b 17,15,13,11
          dc.b 9,8,6,5
          dc.b 3,2,1,0
          dc.b 0,0,0,0
          dc.b 0,0,0,0
          dc.b 0,0,1,2
          dc.b 3,5,6,8
          dc.b 9,11,13,15
          dc.b 17,20,22,24
          dc.b 27,30,32,35
          dc.b 38,41,44,47
          dc.b 50,53,56,59

TONE_TABLE:                            ;16-digit tone table
          dc.w 3792,5381                   ;digit 0 (941, 1335 Hz)
          dc.w 2809,4873                   ;digit 1 (697, 1209 Hz)
          dc.w 2809,5381                   ;digit 2 (697, 1335 Hz)
          dc.w 2809,5953                   ;digit 3 (697, 1477 Hz)
          dc.w 3104,4873                   ;digit 4 (770, 1209 Hz)
          dc.w 3104,5381                   ;digit 5 (770, 1335 Hz)
          dc.w 3104,5953                   ;digit 6 (770, 1477 Hz)
          dc.w 3434,4873                   ;digit 7 (852, 1209 Hz)
          dc.w 3434,5381                   ;digit 8 (852, 1335 Hz)
          dc.w 3434,5953                   ;digit 9 (852, 1477 Hz)
          dc.w 2809,6582                   ;digit A (697, 1633 Hz)
          dc.w 3104,6582                   ;digit B (770, 1633 Hz)
          dc.w 3434,6582                   ;digit C (852, 1633 Hz)
          dc.w 3792,6582                   ;digit D (941, 1633 Hz)
          dc.w 3792,4873                   ;digit * (941, 1209 Hz)
          dc.w 3792,5953                   ;digit # (941, 1477 Hz)
```