Stamp Applications no. 7 (September '95):

DTMF "Touch" Tones are Music to the Ears of this Stamp Transmit/Receive Circuit

Working with the CM8880 DTMF Transceiver, by Scott Edwards

ENCODING and decoding DTMF tones—those musical "touch tones" you hear when you dial the phone—is the current fad in electronics projects. There are gadgets that capture and display or store dialed digits, others that dial stored numbers, and still others that use the hardy dual-tone, multifrequency digits for data transmission or remote control.

Stamp users are an independent lot, so we're going to buck the trend of one-trick DTMF projects. In this edition of Stamp Applications, we'll look at a universal DTMF send/receive solution that can serve as the basis for decoders, loggers, dialers, and even data transceivers.

A Quick Review of DTMF Principles

All the recent excitement over the decades-old DTMF signaling method makes the first part of my job easy. I can breeze through the explanation of the signals themselves with confidence that the curious reader who wants more background can find it his or her recent-magazine stack, for example, *DTMF IR Remote Control System*, *N&V* June '95.

A DTMF signal consists of a mixture of two sinewave tones, often called the *row* and *column* frequencies because of their correspondence to the layout of the phone keypad (figure 1). Engineers at then-mighty Ma Bell chose these tones carefully to ensure that none had a harmonic relationship with the others and that mixing the frequencies would not produce sum or product frequencies that could mimic another valid tone. They specified that the tones be free of distortion, and that the high-group frequencies (the column tones) be slightly louder than the low-group to compensate for the high-frequency rolloff of voice audio systems.

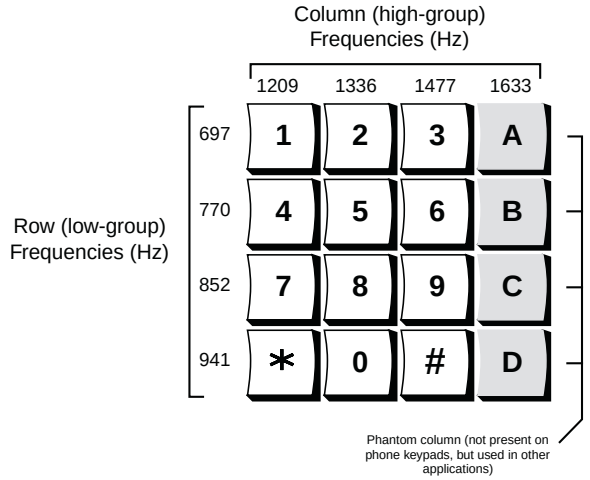

Figure 1. The DTMF tone that's generated when you press a button on the phone keypad consists of a mixture of the row and column frequencies. For example, pressing 9 mixes a column frequency of 1477 Hz with a row frequency of 852 Hz.

Because DTMF was devised in the days before you could hear a pin drop or appreciate Whitney Houston's singing over long distance, it's an exceptionally noise-resistant signaling method. If a properly designed DTMF decoder thinks there's a valid DTMF signal present in some lousy audio, there almost surely is.

DTMF tones can represent one of 16 different states or symbols, as shown in figure 1. That is equivalent to four bits of data, also known as a nibble. Most DTMF decoders can process at least 10 tones per second under the worst of conditions, so DTMF can easily convey 40 bits (5 bytes) of data per second. That's nowhere near the performance of a good communications modem, which can operate nearly 600 times as fast (28,800 bits per second), but it's a lot more robust under noisy line conditions.

Note that the numbers and symbols on the phone keypad don't always match the binary values of the DTMF nibbles. Most notably, the "0" on the keypad is represented in DTMF by a value of 10 (decimal) or 1010 binary. Table 1 summarizes the rest of the four-bit values.

Table 1. DTMF Values, Keypad Symbols

Binary Value	Decimal Value	Keypad Symbol
0000	0	D
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	0
1011	11	*
1100	12	#
1101	13	A
1110	14	B
1111	15	C

A DTMF Transceiver: the CM8880

All you need to generate DTMF tones are two precise, low-distortion sine-wave oscillators and a mixer. To decode DTMF takes just eight tone decoders and a little glue logic. The circuit board to accommodate all this stuff shouldn't be much more than 4 by 6 inches and $50.

Don't like those numbers? Then you will like the California Micro Devices CM8880 Integrated DTMF Transceiver. For $10 or less in single quantity, this chip does everything associated with sending and receiving DTMF tones. The schematic in figure 2 shows a Stamp hookup for both send and receive applications. Listing 1 demonstrates DTMF dialing. Listing 2 shows DTMF reception and display.

Rather than rehash the comments from the program listings, I'd like to show you how to set up and use the CM8880 in your own applications.

Communication with the 8880 takes place over a 4-bit bus, consisting of D0 through D3, with three additional bits selecting modes of operation. Those bits are chip select (CS), read/write (RW) and register select (RS0). Table 2 summarizes the effects of all combinations of CS, RW, and RS0.

To sum up the table, the 8880 is active only when CS is 0. The RW bit determines the data direction; 1 = read (data from 8880 to Stamp) and 0 = write (data from Stamp to 8880). The RS bit determines whether the transaction involves data (DTMF tones) or internal CM8880 functions (instructions or status); 1 = instructions/status and 0 = data.

Once the CM8880 is set up, the Stamp writes 000 to CS, RW, and RS0 to send DTMF; or 010 to read DTMF. Simple as that.

Setting up the CM8880

Before you can use the CM8880, you have to set it up. The device has two control registers, A and B. When you put 001 in bits CS, RW and RS0, the data on D0 through D3 is written to

Table 2. Effects of the CS, RW, and RS0 Bits

CS	RW	RS0	Description
0	0	0	Active: write data (i.e., send DTMF)
0	0	1	Active: write instructions to 8880
0	1	0	Active: read data (i.e., receive DTMF)
0	1	1	Active: read status from 8880
1	0	0	Inactive
1	0	1	Inactive
1	1	0	Inactive
1	1	1	Inactive

Table 3. Functions of Control Register A

Bit	Name	Function
0	Tone Out	0 = tone generator disabled
		1 = tone generator enabled
1	Mode Control	0 = Send and receive DTMF
		1 = Send DTMF, receive call-progress tones
		(DTMF bursts lengthened to 104 ms)
2	Interrupt Enable	0 = Make controller check for DTMF rec'd
		1 = Interrupt controller via pin 13 when DTMF rec'd
3	Register Select	0 = Next instruction write goes to CRA.
		1 = Next instruction write goes to CRB.

Table 4. Functions of Control Register B

Bit	Name	Function
0	Burst	0 = Output DTMF bursts of 52 or 104 ms
		1 = Output DTMF as long as enabled
1	Test	0 = Normal operating mode
		1 = Present test timing bit on pin 13
2	Single/ Dual	0 = Output dual (real DTMF) tones.
		1 = Output separate row or column tones
3	Column/ Row	0 = If above = 1, select row tone.
		1 = If above = 1, select column tone.

control register A (CRA). Table 3 summarizes the functions of the four bits of CRA.

Looking at listing 1, the dialer, we see that 1011 was written to CRA. That works out to (right to left) 1 (tone generator on), 1 (Send DTMF, long tones), 0 (don't generate interrupts), and 1 (next write to CRB). Listing 2 writes 1000 to CRA: 0 (tone-generator off), 0 (receive DTMF), 0 (don't generate interrupts) and 1 (next write to CRB).

Although we don't use it in our applications, the 8880 has a call-progress detection feature, which can be enabled via bit 1 of CRA. Call-progress is a collective term for the dial tone, busy signal, and ringing signal. These tones are all around 400 Hz. The CM8880's call-progress

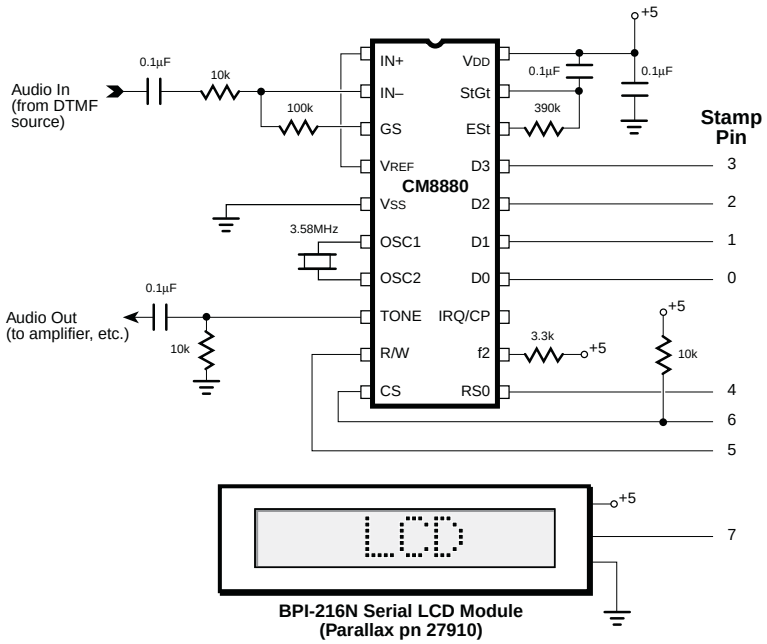

Figure 2. Schematic diagram for DTMF send and receive applications.

filter accepts audio from the normal input and passes frequencies from 300 to 500 Hz to the IRQ/CP pin. You could add hardware to further filter and detect these tones.

Since both listings also set up register B (CRB), take a look at table 4 to see how it works.

Listing 1 (dialer) writes 0s to all bits of CRB, meaning burst mode, normal operation, real DTMF, and (if DTMF weren't selected) row tones. Listing 2 (decoder) also clears CRB. Although CRB has little or nothing to do with DTMF reception, it's wise to always initialize it to a known state.

A Few Hardware Notes

To hear the DTMF tones generated by the CM8880, I connected a small speaker/amplifier (Radio Shack part no. 277-1008C) to the audio output. The amplifier's high gain and the 8880's healthy audio output quickly taught me to set the volume control low! I was able to dial the phone by holding the speaker close to the mouthpiece.

I used two approaches to generate DTMF test tones for the decoder application. The first was to connect an old touch-tone phone to a 9V battery and the audio-input coupling capacitor (0.1µF shown in figure 2) to the positive terminal. The polarity of the connection didn't matter, as phones are designed to tolerate wiring mixups. However, this setup sucked better than 50 mA from the battery.

NOTE:

Obsolete information deleted.

DTMF and a glass of red...

It's almost embarrassing to admit that that's all there is to designing DTMF send/receive applications based on the CM8880 and the Stamp. I deliberately skirted issues having to do with direct connection to the phone line, since that's a subject worthy of two (or three!) articles of this size. Your best bet is to see how others have done it successfully, or to purchase a

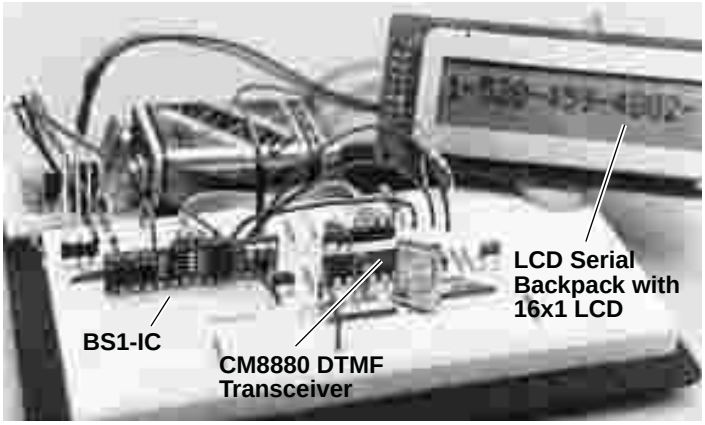

Figure 3. DTMF send/receive setup.

commercially made Data Access Arrangement (DAA) to serve as an interface. One manufacturer of DAAs is Cermetek, Sunnyvale, CA, 408-752-5000.

Don't limit yourself to phone applications. Remember what I said about DTMF serving as a reliable data-transfer method? You could connect CM8880-equipped Stamps to inexpensive two-way radios to create a wireless network for data acquisition over a 1/4-mile radius.

I experimented briefly with this idea, using the DTMF output from the amplified speaker to trigger the voice-activated transmit feature of a Radio Shack headset walkie-talkie (part no 21-406). The voice activation was fast enough that even the first tone got through. This means that you wouldn't need an additional pin to switch between transmit and receive.

This is an application that a lot of Stamp users have been clamoring for. I remember in particular that grape growers need to gather and evaluate "microclimate" data regarding temperature and humidity among their pampered vines, and would kill for a wireless way to obtain it. Here it is: Cheers. (And I prefer red, if you please, a Cabernet Sauvignon or Merlot.)

NOTE:

Obsolete information deleted.

Sources

NOTE:

Parallax (www.parallaxinc.com) carries the CM8880 chip both separately and as part of an App Kit (pn 27915) that includes printed documentation and program examples on disk for the Stamp I and Stamp II.

This article was originally published in 1995. The Stamp Applications column continues with a changing roster of writers. See www.nutsvolts.com or www.parallaxinc.com for current Stamp-oriented information.

' Listing 1. Stamp-Based Autodialer

```
' Program: DIAL.SRC (Sends a string of DTMF tones via the 8880)
' This program demonstrates how to use the CM8880 as a DTMF tone
' generator. All that's required is to initialize the 8880 properly,
' then write the number of the desired DTMF tone to the 8880's
' 4-bit bus.

' The symbols below are the pin numbers to which the 8880's
' control inputs are connected, and one variable used to read
' digits out of a lookup table.

SYMBOL      RS_p = 4          ' Register-select pin (0=data).
SYMBOL      RW_p = 5          ' Read/Write pin (0=write).
SYMBOL      CS_p = 6          ' Chip-select pin (0=active).
SYMBOL      digit = b2        ' Index of digits to dial.

' This code initializes the 8880 for dialing by writing to its
' internal control registers CRA and CRB. The write occurs when
' CS (pin 6) is taken low, then returned high. See the accompanying
' article for an explanation of the 8880's registers.

let pins = 255          ' All pins high to deselect 8880.
let dirs = 255          ' Set up to write to 8880 (all outputs).
let pins = %00011011    ' Set up  register A, next write to register B.
high CS_p
let pins = %00010000    ' Clear register B; ready to send DTMF.
high CS_p
```

```
' This for/next loop dials the seven digits of my fax number. For
' simplicity, it writes the digit to be dialed directly to the output
' pins. Since valid digits are between 0 and 15, this also takes RS,
' RW, and CS low--perfect for writing data to the 8880. To complete
' the write, the CS line is returned high. The initialization above
' sets the 8880 for tone bursts of 200 ms duration, so we pause
' 250 ms between digits. Note: in the DTMF code as used by the phone
' system, zero is represented by ten (1010 binary) not 0. That's why
' the phone number 459-0623 is coded 4,5,9,10,6,2,3.

for digit = 0 to 6
  lookup digit,(4,5,9,10,6,2,3),pins ' Write current digit to pins.
  high CS_p                  ' Done with write.
  pause 250                  ' Wait to dial next digit.
next digit
end
```

' Listing 2. DTMF Decoder and Display

```
' Program: DTMF_RCV.BAS (Receives and display DTMF tones using the 8880)
' This program demonstrates how to use the 8880 as a DTMF decoder. As
' each new DTMF digit is received, it is displayed on an LCD Serial
' Backpack screen. If no tones are received within a period of time
' set by sp_time, the program prints a space (or other selected character)
' to the LCD to record the delay. When the display reaches the righthand
' edge of the screen, it clears the LCD and starts over at the left edge.

SYMBOL      RS_p = 4          ' Register-select pin (0=data).
SYMBOL      RW_p = 5          ' Read/Write pin (0=write).
SYMBOL      CS_p = 6          ' Chip-select pin (0=active).
SYMBOL      dtmf = b2         ' Received DTMF digit.
SYMBOL      dt_Flag = bit0    ' DTMF-received flag.
SYMBOL      home_Flag = bit1  ' Flag: 0 = cursor at left edge of LCD.
SYMBOL      polls = w2        ' Number of unsuccessful polls of DTMF.
SYMBOL      LCDw = 16         ' Width of LCD screen.
SYMBOL      LCDcol = b3       ' Current column of LCD screen for wrap.
SYMBOL      LCDcls = 1        ' LCD clear-screen command.
SYMBOL      I = 254           ' LCD instruction toggle.
SYMBOL      sp_time = 1000    ' Print space this # of polls w/o DTMF.

' This code initializes the 8880 for receiving by writing to its
' internal control registers CRA and CRB. The write occurs when
' CS (pin 6) is taken low, then returned high.

let pins = %01111111    ' Pin 7 (LCD) low, pins 0 through 6 high.
let dirs = %11111111    ' Set up to write to 8880 (all outputs).
let pins = %00011000    ' Set up register A, next write to register B.
high CS_p
let pins = %00010000    ' Clear register B.
high CS_p
let dirs = %11110000    ' Now make set the 4-bit bus to input.
high RW_p               ' And set RW to "read."
serout 7,n2400,(I,LCDcls)   ' Clear the LCD screen.

' In the loop below, the program checks the 8880's status register
' to determine whether a DTMF tone has been received (indicated by
' a '1' in bit 2). If no tone, the program loops back and checks
```

```
' again. If a tone is present, the program switches from status to
' data (RS low) and gets the value (0-15) of the tone. This
' automatically resets the 8880's status flag.
again:
  high RS_p             ' Read status register.
  low CS_p              ' Activate the 8880.
  let dt_flag = pin2    ' Store status bit 2 into flag.
  high CS_p             ' End the read.
if dt_Flag = 1 then skip1     ' If tone detected, continue.
let polls = polls+1           ' Another poll without DTMF tone.
if polls < sp_time then again ' If not time to print a space, poll again.
if LCDcol = LCDw then skip2   ' Don't erase the screen to print spaces.
let dtmf = 16                 ' Tell display routine to print a space.
gosub Display                 ' Print space to LCD.
skip2:
let polls = 0           ' Clear the counter.
goto again              ' Poll some more.
skip1:                  ' Tone detected:
let polls = 0           ' Clear the poll counter.
low RS_p                ' Get the DTMF data.
low CS_p                ' Activate 8880.
let dtmf = pins & %00001111   ' Strip off upper 4 bits using AND.
high CS_p               ' Deactivate 8880.
gosub display           ' Display the data.
goto again              ' Do it all again.

Display:
if LCDcol < LCDw then skip3   ' If not at end of LCD, don't clear screen.
serout 7,n2400,(I,LCDcls)   ' Clear the LCD screen.
let LCDcol = 0                ' And reset the column counter.
skip3:                        ' Look up the symbol for the digit.
if LCDcol=0 AND dtmf=16 then ret    ' No spaces at first column.
lookup dtmf,("D1234567890*#ABC-"),dtmf
serout 7,n2400,(dtmf)         ' Write it to the Backpack display.
let LCDcol = LCDcol + 1       ' Increment the column counter.
ret:
return
```