Analog – Sine Wave Generation with PSoC® (Demonstration with CTCSS)

AN2025

Author: Jeff Dahlin

Associated Project: Yes
Associated Part Family: CY8C23x33, CY8C24x23A, CY8C24x94
CY8C27x43, CY8C28xxx, CY8C29x66
Software Version: NA
Associated Application Notes: None

Application Note Abstract

This application note demonstrates how to implement a Continuous Tone Coded Squelch System (CTCSS) carrier generator in PSoC® 1. The implementation uses a 256-byte sine wave lookup table in ROM and a 6-bit Voltage Output Digital to Analog Convertor (DAC6) User Module (UM). The basic routine to generate the sine wave is not application specific and can be used to generate other waveforms and frequencies.

Introduction

Continuous Tone Coded Squelch System (CTCSS) or similar mechanisms are used in most handheld radios to allow multiple users to share one carrier frequency. The goal of CTCSS is to allow the receiving radio to suppress (or squelch) signals that are not intended for its user. If a pair of radios is set to the same CTCSS tone, then the audio transmitted by one radio is received on the other radio's speaker.

CTCSS works by mixing a single tone with the transmitted voice audio at all times. A radio receiving a signal checks if the CTCSS tone selected for that radio is present. If the tone is present, the receiving radio outputs the voice audio. If the tone is not present, any signal received is not sent to the speaker. This allows multiple radios to coexist on the same carrier frequency in the same area without users having to listen to everyone.

CTCSS Carrier Frequencies

CTCSS uses 38 different frequencies between 67.0 Hz and 250.3 Hz as the selection tones. Table 1 lists the frequency associated with each of the tones. Some radios use additional "nonstandard" frequencies for CTCSS functionality. This application only uses the standard CTCSS frequencies.

Table 1. Standard CTCSS Frequencies

Tone	Frequency	Tone	Frequency
1	67.0 Hz	20	131.8 Hz
2	71.9 Hz	21	136.5 Hz
3	74.4 Hz	22	141.3 Hz
4	77.0 Hz	23	146.2 Hz
5	79.7 Hz	24	151.4 Hz
6	82.5 Hz	25	156.7 Hz
7	85.4 Hz	26	162.2 Hz
8	88.5 Hz	27	167.9 Hz
9	91.5 Hz	28	173.8 Hz
10	94.8 Hz	29	179.9 Hz
11	97.4 Hz	30	186.2 Hz
12	100.0 Hz	31	192.8 Hz
13	103.5 Hz	32	203.5 Hz
14	107.2 Hz	33	210.7 Hz
15	110.9 Hz	34	218.1 Hz
16	114.8 Hz	35	225.7 Hz
17	118.8 Hz	36	233.6 Hz
18	123.0 Hz	37	241.8 Hz
19	127.3 Hz	38	250.3 Hz

Frequency Generation - Implementation

In the project accompanying this application note, the CTCSS frequencies are generated by transferring data from a 256-entry, 8-bit Look up Table (LUT) into a DAC6 UM at a fixed rate. The DAC is updated at a constant rate in an Interrupt Service Routine (ISR) that is controlled by an 8-bit Timer (Timer8 UM). The frequency of the CTCSS tone is adjusted by varying the step size through the LUT.

The 256-byte LUT contains data that represents one cycle of a sine wave. This data is stored in sign and magnitude format. Sign and magnitude is the native format of the DAC data register. Write to the DAC is more efficient in this format.

In the example project, the Timer8 is set to approximately 10 kHz. The frequency of the output waveform is determined by the step size of the index through the LUT.

For example, for a 10 kHz update rate and a step size of '1', the resulting sine wave has a frequency of about 39 Hz (10,000/256). If a step size of '2' is used, the resulting sine wave has a frequency of twice that, at about 78 Hz. The Excel file used to generate the LUT is provided along with this application note. It contains the options for the clock frequency and the number of steps in LUT, and generates the LUT based on them.

The 256-byte LUT is stored in ROM and is accessed using the MCU's `INDEX` instruction. The `INDEX` instruction uses a base address, which is hardcoded as the operand in the instruction and an index, which is the value in the Accumulator at the start of the command. When the `INDEX` instruction is executed, the table entry pointed to by the sum of the base address and the index is loaded into the Accumulator. The 8-bit index for the LUT comes from the upper byte of a 16-bit index (`iCTCSSFreqIndex` in the project). The upper byte can be considered as the integer portion of the index and the lower byte as the fractional portion of the index. To step through the LUT, a 16-bit index increment (`iCTCSSFreqInc` in the project), also having an integer portion and a fractional portion, is added to the 16-bit accumulated index. This allows the use of fractional increments, which creates a more accurate frequency than is possible with an integer increment.

In the example described earlier, where a step size of '2' resulted in an output frequency of 78 Hz, the index increment is 0x0200. To get an output frequency of 67 Hz (CTCSS tone 1), a step size of 1.72 is needed. The index increment in this case is 0x01b3.

Frequency Selection

In this project, a function is provided that sets the index increment value for the desired frequency. This function, `SetCTCSSFreq()`, is passed a 1-byte argument that is the CTCSS tone number. The tone number is manipulated (subtract 1 and multiply by 2) to convert it into an index for a 38 entry 16-bit LUT which contains index increment values.

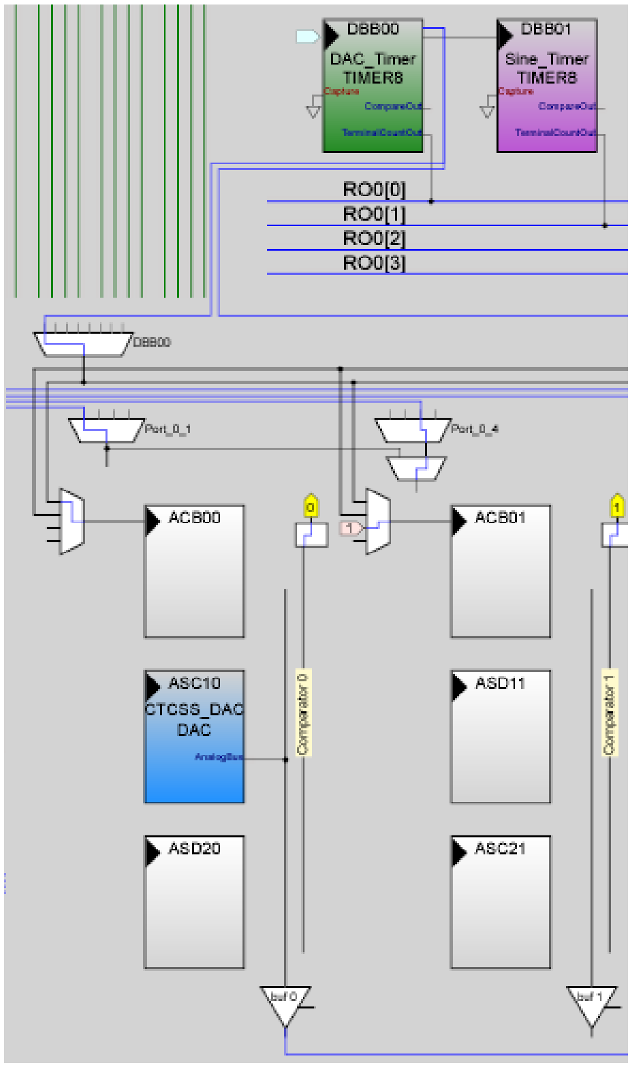
Phase Coherence

A side effect of the way in which the output frequency is changed (the accumulated index variable is not cleared) is that the sine wave exhibits no discontinuity when the frequency changes. Although this is not critical for CTCSS generation, it may be required in other waveform applications.

Hardware Configuration

Figure 1 shows the placement of the three user modules used in this project. CTCSS_DAC, a DAC6, outputs the analog signal that is the sine wave. CTCSS_DAC is configured to use a SignAndMagnitude data format. The output to the analog column bus is enabled and Buf0 is enabled to output the analog signal to Port0[3].

Figure 1. CTCSS User Module Placement



DAC_Timer, a Timer8 configured to divide the 48M clock by 61, is used to generate the column clock for the DAC6. This results in a DAC update rate of 197 kHz. A high DAC update rate is required for this application. The DAC update frequency appears in the spectrum of the output. The higher the DAC update rate, the more effectively it is removed with a simple R-C low pass filter.

Sine_Timer, a Timer8 configured to divide the output of DAC_Timer by 78, generates an output frequency of 10.088 kHz.

Port0[3] (pin 3 on a 28-pin package) is set to AnalogOutBuf0 with a drive mode of High Z to enable the analog signal to be output.

The frequency accuracy requirement for CTCSS is usually better than the ±2.5% accuracy of the Internal Main Oscillator in the PSoC® microcontroller; therefore an external crystal oscillator is required. The Port1[0] and Port1[1] pins are set to High Z and a 32.678 kHz crystal is connected to the part as specified in the Cypress application note AN2027, *Using the PSoC Microcontroller External Crystal Oscillator.*

Note To check working of the project without an external oscillator, the PLL_Mode in global parameters must be set to “Disable” and the 32K_Select must be set to “Internal”. If a crystal is added, but the selection for the 32K_Select is external, the ECO circuit continues to oscillate. The frequency of this oscillation is far from 32 kHz. With these settings, the project outputs at an unexpected frequency.

ISR Overhead

If the PSoC microprocessor also has other tasks, the amount of CPU overhead used by the ISR while generating the CTCSS output is important. There are three factors that impact the ISR overhead. They are the DAC6 analog column clock speed, CPU clock speed, and Sine_Timer clock rate.

The Sine_Timer ISR involves a write stall when updating the CTCSS_DAC. A write stall pauses the CPU until the start of the rising edge of the Phi1 clock in the analog column. This is done to prevent glitches from occurring on the DAC output. (See the Analog Synchronization Interface section in the Analog Interface chapter of the Technical Reference Manual (TRM) for more details on write stall.) The worst-case stall period for the DAC_Timer period used in this project is 5.1 µsec. Apart from the write stall, the ISR takes an additional 93 CPU clock cycles to execute.

With a DAC update rate of 10.088 kHz, the worst-case ISR overhead is 15%. Table 2 shows the ISR overhead for different CPU clock rates, assuming an analog column clock of 197 kHz and a DAC update rate of 10.088 kHz.

Table 2. CTCSS ISR Overhead

CPU_Clock	ISR Overhead
24 MHz	15 %
12 MHz	24 %
6 MHz	44 %
3 MHz	82 %
This project cannot be run below 3 MHz	

If different timings are used for this application, the resulting ISR overhead differs from those listed in Table 2.

Tradeoffs

There are some tradeoffs that must be made when designing a waveform generator, as discussed in this section.

ISR overhead versus DAC update rate is one tradeoff that must be considered. More steps in a wavelength results in a lower harmonic distortion of the waveform. Fewer steps in one wavelength reduce the ISR overhead, leaving more of the CPU available for other tasks.

Waveform distortion versus output-filter complexity is another part of the same tradeoff. If the DAC update rate is much higher than the frequency of the output waveform, the major contributor to distortion is at the DAC update frequency. In this case a simple R-C low pass filter on the output (often called a reconstruction filter) can acceptably reduce the waveform distortion. If the DAC update rate is too close to the frequency of the output waveform, the harmonics of the output frequency is the major contributor to distortion. In this case something more than a simple R-C low pass filter is needed to clean up the waveform. A Low Pass Filter User Module can be used.

In some designs, due to limited resources, the source for the DAC’s analog column clock must be shared with another function. For this, a slower clock must be used. This affects the ISR overhead; it increases the maximum stall time, which in turn increases the waveform distortion by moving the DAC clock frequency closer to the output frequency. Ideally, a frequency close to the DAC6 maximum update rate is used.

In this project, a 6-bit DAC is selected over an 8-bit DAC. The 8-bit DAC results in more precise steps, but uses more analog SoC blocks. Experiments show that the number of steps per waveform has a much greater impact on the accuracy of the output waveform than the DAC resolution.

Performance

After configuring the timers as described in the previous sections and using an external crystal, data was collected to determine the frequency accuracy and the waveform distortion of the output.

The output frequencies were measured for all 48 CTCSS tones using a Fluke 87 Digital Multimeter. The results of these measurements are shown in Table 3. The range of the frequency error is between +0.10% and –0.13%.

The waveform distortions for selected CTCSS tones (1, 7, 13, 19, 26, 32, and 38) are calculated based on measurements taken with a Hewlett-Packard 3585A Spectrum Analyzer. The measurements are taken directly from the output of the PSoC microcontroller with no reconstruction filter. The results are graphed in Figure 2. The maximum distortion, 3.8%, occurred at the highest output frequency, as expected.

Additional testing was run using a DAC update rate (Sine_Timer) of 6.5 kHz. The maximum distortion increased to 6.3%.

Figure 2. Waveform Distortion

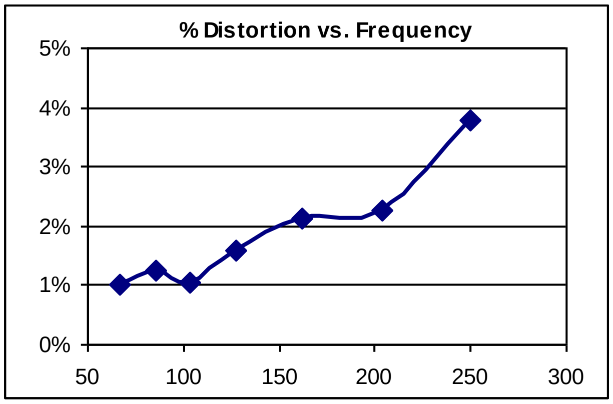


Table 3. Frequency Accuracy

Tone #	Frequency	Measured	Error	Tone #	Frequency	Measured	Error
1	67.0	67.09	-0.13%	20	131.8	131.83	-0.02%
2	71.9	71.83	0.10%	21	136.5	136.47	0.02%
3	74.4	74.45	-0.07%	22	141.3	141.20	0.07%
4	77.0	77.08	-0.10%	23	146.2	146.16	0.03%
5	79.7	79.71	-0.01%	24	151.4	151.37	0.02%
6	82.5	82.45	0.06%	25	156.7	156.79	-0.06%
7	85.4	85.39	0.01%	26	162.2	162.15	0.03%
8	88.5	88.50	0.00%	27	167.9	167.87	0.02%
9	91.5	91.54	-0.04%	28	173.8	173.83	-0.02%
10	94.8	94.81	-0.01%	29	179.9	179.81	0.05%
11	97.4	97.38	0.02%	30	186.2	186.18	0.01%
12	100.0	100.01	-0.01%	31	192.8	192.72	0.04%
13	103.5	103.51	-0.01%	32	203.5	203.40	0.05%
14	107.2	107.22	-0.02%	33	210.7	210.70	0.00%
15	110.9	110.93	-0.03%	34	218.1	218.10	0.00%
16	114.8	114.78	0.02%	35	225.7	225.60	0.04%
17	118.8	118.83	-0.03%	36	233.6	233.50	0.04%
18	123.0	123.12	-0.10%	37	241.8	242.10	-0.12%
19	127.3	127.22	0.06%	38	250.3	250.30	0.00%

Figure 3 and Figure 4 show the waveforms for Tone 1 and Tone 38, respectively. They are captured with a Tektronix TDS 3034 Digital Storage Oscilloscope. Notice that the individual DAC increments are more apparent at the higher frequency. This is because there are fewer steps per cycle at higher frequencies. This results in a greater waveform distortion.

Figure 3. Tone 1 (67.0 Hz) Waveform

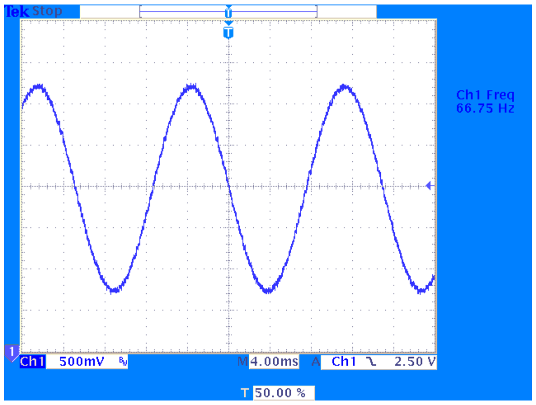
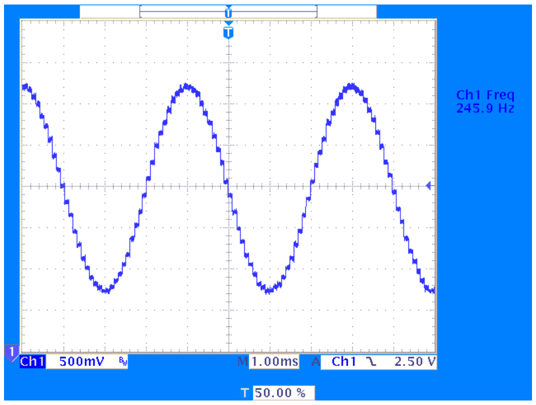


Figure 4. Tone 38 (250.3 Hz) Waveform



Document History

Document Title: Analog – Sine Wave Generation with PSoC® (Demonstration with CTCSS)

Document Number: 001-40881

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1536344	JVY	See ECN	New application note
*A	2793434	YARA	10/27/09	Updated part number from CY8C25xxx to CY8C27xxx. Updated accompanying projects from PSoC Designer version 2.1 to 5.0. Updated document text where appropriate.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/