# Direct Waveform Synthesis for Software Radios


Matthew L Welborn
MIT Lab for Computer Science
mwelborn@lcs.mit.edu



*Abstract*— **This work presents results that show how direct digital synthesis (DDS) techniques can be extended to produce frequency modulated (FM) waveforms as well as many types of digitally modulated waveforms, such as PSK or QAM. We show how digitally modulated waveforms can be efficiently synthesized from tables of pre-computed samples. This work also reports on the implementation of these algorithms in a software radio system.**


## I. Introduction

The technique of direct digital synthesis (DDS) is gaining popularity as a method of generating sinusoidal signals in digital systems [5]. This technique is especially well suited to software radio systems implemented on architectures which use general purpose processors (GPPs) because these systems often have large amounts of main memory available for storage of pre-computed output samples. These systems can therefore provide great flexibility in the type of waveforms which can be generated. The algorithms developed below show how DDS techniques can be extended from the simple generation of sinusoids to produce FM or even digital modulation waveforms.

## II. FM Waveforms

We first review how to use a pre-computed table of samples of an *oversampled* sinusoid to generate a sinusoid of effectively arbitrary frequency. Our table $T$ is a table containing $2^N$ pre-computed output samples and $T[i]$ is the $i$th element in the table. We fill our table with $2^N$ uniformly-spaced samples of a single period of a sinusoid:

$$T[i] = \cos\left[2\pi\left(\frac{i}{2^N}\right)\right], \text{ for } i = 0, 1, ..., 2^N - 1 \tag{1}$$

We can now use this table to generate any sinusoid with a frequency $f_c = \frac{kf_s}{2^N}$, where $f_s$ is the sample frequency in hertz and $k \in \{1, ..., 2^N - 1\}$. To do this we use a *phase accumulator* to keep track of the instantaneous phase of the sinusoid [2]. Because this sinusoid has a constant frequency, at each time step the instantaneous phase is incremented by a constant $\left(\frac{2\pi f_c}{f_s}\right)$ radians. The desired sinusoid, $s[n]$, is then:

$$\begin{aligned} s[n] &= \cos(2\pi f_c n T_s) \\ &= T\left[\left(\sum_{i=1}^{n} \Delta\right)(mod\ 2^N)\right] \text{ where } \Delta = \frac{2^N f_c}{f_s} \end{aligned} \tag{2}$$

Here $T_s = \frac{1}{f_s}$ is the sample time and we have used the the fact that the sinusoid is periodic (with period equal to $2^N$ samples) to ensure that the contents of the phase accumulator point directly to the correct sample in table $T$. The $(mod\ 2^N)$ function can be performed automatically if we use an $N$-bit register (or the top $N$-bits of a longer register) to store the phase accumulator.

We can also use this phase accumulator method to perform frequency modulation. The discrete-time equation for the instantaneous frequency, $f_i[n]$ of an FM signal (with carrier frequency $f_c$ and modulating signal $m[n]$) is $f_i[n] = K_f m[n] + f_c$. Here $K_f$ is the modulation constant for the signal which depends on the desired maximum frequency deviation of the FM signal and the amplitude of the input signal, $m[n]$. We perform discrete-time integration of this to get the instantaneous phase of the constant-amplitude FM signal. The resulting FM signal is then:

$$s[n] = \cos(2\pi T_s \sum_{i=0}^{n}(K_f m[i] + f_c)) \tag{3}$$

If we write this in terms of the instantaneous phase increment at each step, $\Delta_i$, we can generate the output signal directly using the phase accumulator and table look-ups:

$$s[n] = \cos\left(\frac{2\pi}{2^N}\left[\left(\sum_{i=0}^{n}\Delta_i\right)\right]\right)$$
$$= T\left[\left(\sum_{i=0}^{n}\Delta_i\right)(mod\ 2^N)\right] \quad (4)$$

where

$$\Delta_i = \left(\frac{2^N K_f}{f_s}\right)m[i] + \left(\frac{2^N f_c}{f_s}\right) = C_1 m[i] + C_2 \quad (5)$$

In (4) we see that the index value required to look up each output sample in the table is simply the value of the phase accumulator which accumulates a varying phase increment. In (5) we see that this phase increment can be computed from the samples of the input modulating signal with only one multiplication and addition operation if we pre-compute the constants $C_1$ and $C_2$ as shown. We also note here that the FM signal can be generated with any carrier frequency, $f_c$, that meets the same conditions required for the sinusoid generator in (2).

## III. Digitally Modulated Waveforms

Our work has also included the development of algorithms which can generate digital modulation waveforms from tables of pre-computed output samples. Recall that to produce an FM signal we selected individual samples from a pre-computed table; to produce digital waveforms, however, we do not use the phase accumulator technique, but rather we pre-compute *symbol-length sequences* of output samples and then generate the continuous output waveform by piecing together the pre-computed sequences of samples.

To understand how this is accomplished, we start with the discrete-time equation for a general pulse-amplitude modulated (PAM) with symbol values $A_k$ taken from the set $\mathcal{A}$ of symbol values:

$$s[n] = \sum_{k=-\infty}^{\infty} A_k p[n - kN_s] \quad (6)$$

Here $p[n]$ is the unit-pulse response of the discrete-time transmit filter and $N_s$ is the over-sampling ratio, or the number of samples generated for each symbol period $T$. A typical digital hardware system which might implement this equation is shown in Fig. 1. The figure shows the modulation operation for complex-valued symbols as well as a frequency translation to some intermediate carrier frequency in discrete-time. To simplify implementation, we will approximate the general response $p[n]$ with a time-windowed version, $p_z[n]$, which is the same response symmetrically truncated to have a length of $K$ symbol-periods, or $KN_s$ samples. After truncations, we can write $p_z[n]$ as

$$p_z[n] = \begin{cases} p[n] & |n| \le \frac{KN_s}{2} \\ 0 & |n| > \frac{KN_s}{2} \end{cases} \quad (7)$$

We can see an example of this approximation in Fig. 2, where a sampled raised-cosine pulse has been truncated to $K = 6$ symbol periods in length. Also indicated in Fig. 2 are the sets $\phi_1, ..., \phi_K$ of samples corresponding to each symbol period of the truncated pulse $p_z[n]$. We will use these sets of samples to pre-compute all of the possible symbol-length sequences that make up the approximated

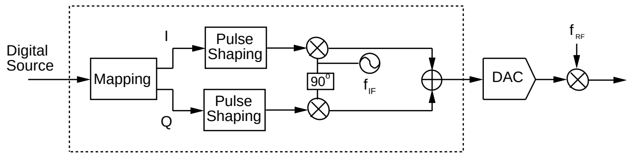


Fig. 1. Block diagram of a general modulator for PAM with complex-valued data symbols.

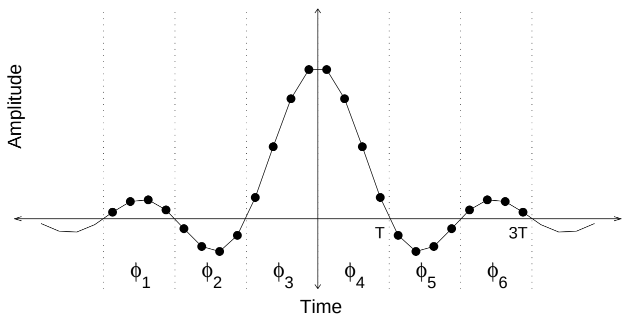


Fig. 2. Example showing the samples of the discrete pulse $p_z[n]$, truncated to $K = 6$ symbol periods with $N_s = 4$ samples per symbol.

output waveform, $\hat{s}[n]$:

$$\hat{s}[n] = \sum_{k=-\infty}^{\infty} A_k p_z[n - kN_s]$$

$$= \sum_{k=k_n-\frac{K}{2}}^{k_n+\frac{K}{2}} A_k p_z[n - kN_s] \quad (8)$$

In (8) we have limited the index of summation to only those $k$ for which $p_z[n]$ is non-zero, that is the $K$ symbols in the region centered around symbol $A_{k_n}$ where $k_n = \text{floor}(n/N_s)$. Because the length of the truncated pulse-shaping filter response is $K$ symbol-periods, we see that each output sample depends on exactly $K$ data symbols, and that all of the samples in each symbol-length sequence of output samples depend on the *same* $K$ data symbols. We will denote these symbol-length sequences of output samples as $S_k$ and we see that these sequences of $N_s$ samples are simply sums of the sequences $\phi_1, ..., \phi_K$, each multiplied by the appropriate symbol values:

$$S_k = A_k \phi_K + A_{k-1}\phi_{K-1} + ... + A_{k-K}\phi_1 \quad (9)$$

If we assume that there are $|\mathcal{A}| = 2^B$ different data symbols, then there are $2^{BK}$ different possible output sequences $S_k$ (each a sequence of $N_s$ samples). We compute all of these possible sequences, storing the results in a table in memory. When we need to generate an output waveform, we can then quickly retrieve the required sequence of samples from the table. Fig. 3 shows a block diagram of the table-based digital modulator showing the steps required to produce the output waveform: (1) compute the table index based on input data bits and (2) retrieve the required output samples.

In the following sections, we address the effects of truncating the transmit filter response as well as the dependence of look-up table size upon both the filter truncation length and the symbol constellation size.

## IV. Implementation Issues

In the following sections, we address the effects of truncating the transmit filter response as well as the dependence of look-up table size upon both the filter truncation length and the symbol constellation size.

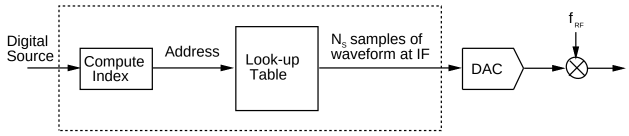


Fig. 3. Block diagram of the proposed table-based modulator for PAM.

### A. *Effects of truncating the filter response*

In the previous section, we proposed truncating the unit-pulse response of the pulse-shaping filter in order to simplify the computation of output waveforms. In practice, this type of truncation occurs in any system which uses a finite impulse response (FIR) filter to implement the pulse shaping filter in the transmitter. One of the effects of truncating this response in the time-domain is to spread the spectrum of the output signal in the frequency domain, so K must be selected with this in mind. To see this effect more clearly, we will show the effect of this truncation on the spectrum of the transmit pulse and the power spectrum density of the resulting PAM signal. For a general pulse-shaping filter response, $p[n]$, truncating the response is equivalent of windowing (multiplying) the response with a rectangular window of width $KN_sTs$. In the frequency domain, this effect of this windowing is a convolution of the pulse spectrum, $P(f)$, with the spectrum of the windowing function, $W(f)$:

$$P_z(f) = P(f) * W(f) \quad (10)$$

The spectra of an example pulse and windowing function are are shown in Fig. 4. Here the pulse is an ideal low-pass pulse and the window has a width of $K = 6$ symbol periods in the time domain. Because these spectra are the transforms of discrete-time sequences, they are periodic with period $f_s = 1/T_s$ in the frequency domain, but only a single period for shown here for clarity. The spectrum of the *truncated* pulse can be computed as the

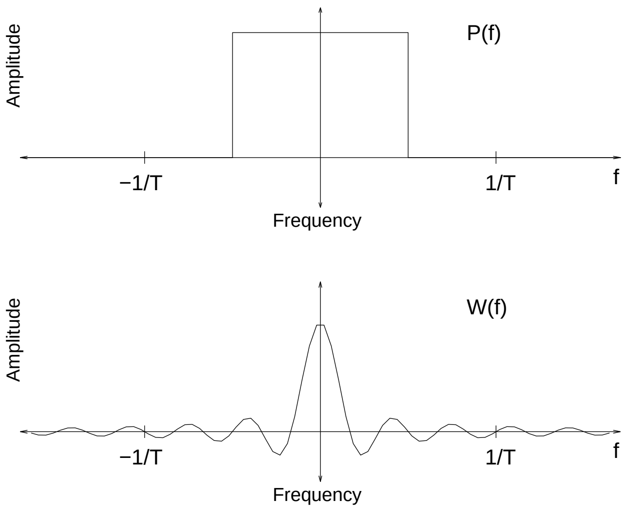


Fig. 4. Plot of transforms of ideal pulse, $P(f)$, and windowing function, $W(f)$, for a rectangular window of $K = 6$ symbol periods.

periodic convolution as in (10):

$$P_z(f) = \int_{-\frac{1}{2T_s}}^{\frac{1}{2T_s}} P(\theta)W(f-\theta)d\theta$$
$$= \int_{-\frac{1}{2T}}^{\frac{1}{2T}} W(f-\theta)d\theta = \int_{-\frac{1}{2T}}^{\frac{1}{2T}} \frac{\sin(\pi(f-\theta)T)}{\pi(f-\theta)T}d\theta \tag{11}$$

This integral in (11) does not have a closed form solution, but in Fig. 5 we can see the result for the example pulse and window functions shown in Fig. 4. Here we see that the result of truncation is to spread the initially ideal pulse shape, $P(f)$, by an amount approximately equal to the the width of $W(f)$. Also, this truncation has introduced sidebands in the spectrum of the pulse. The amount of spreading and the amplitude of the sidebands can be controlled by changing the width of the window function ($K$ symbols) or by using a non-rectangular window function commonly used in FIR filter design [3].

The effect of the truncation on the power spectrum density of the final modulated signal is similar to the effect on the pulse spectrum. If we assume that the data symbol sequence is zero-mean and the symbols are random and uncorrelated, then the PSD, $S_{\hat{s}}(f)$, of the continuous-time version of the modulated signal in (8) is given by [4]:

$$S_{\hat{s}}(f) = \frac{\sigma_A^2}{T}|P_z(f)|^2 \tag{12}$$

where $\sigma_A^2$ is the variance of the random symbol sequence. From this we see that the PSD of the modulated signal will be spread in a similar manner to the original pulse spectrum by the time-domain truncation.

### *B. Size of the sample sequence tables*

As discussed above, there will be $2^{BK}$ different possible output sequences of samples that must be pre-computed and stored. The total size of our look-up table is therefore $2^{BK} \times N_s \times W$ bytes where there are $|\mathcal{A}| = 2^B$ symbols in the constellation and we use W-byte length samples. For example, a 4-PAM system with a 6-symbol length pulse-shaping transmit filter generating 10 samples per symbol and 2 bytes/sample (16 bits) requires a table size of 80 kilobytes (KB), which is quite reasonable and would actually fit in the *cache* on most modern GPPs. For larger symbol constellations or longer filter responses, the table size grows rapidly (exponentially in the length of the filter response or number of bits/symbol). One way to limit this growth is to use several tables in which we perform parallel look-ups and simply add the pre-computed values. To use this technique we must find an appropriate way to decompose the required sample sequences.

One solution is to decompose the table into $D$ separate tables by having the index of each table depend only on $\frac{K}{D}$ consecutive symbols instead of the full $K$. in this case the total table space is

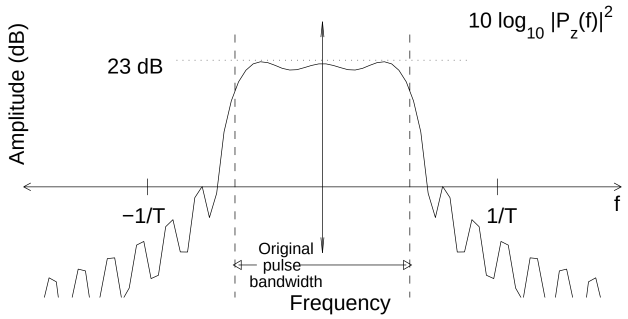


Fig. 5. Plot of the transform of truncated pulse, $P_z(f)$, for a rectangular window of $K = 6$ symbol periods.

$D \times 2^{(BK/D)} \times N_s \times W$ bytes and the cost of this decomposition is that we perform $D$ look-ups and $(D-1)$ additions. Using this technique, the 4-PAM example above would require only $2 \times 2^{(2\times6/2)} \times 10 \times 2 \approx 2.5$ KB if we decompose the original table into two tables. A larger constellation example, such as 16-QAM, with $K = 6$ length filter response would require $3 \times 2^{(4\times6/3)} \times 10 \times 2 \approx 15$ KB using a decomposition in to three tables (as opposed to over 300 megabytes using a single table) while requiring two addition operations per output sample.

This technique of pre-computing output sample sequences instead of direct computation at each step is especially helpful when we are generating an output waveform which might contain multiple signals combined using frequency multiplexing, as might be done in a wideband cellular or PCS-type transmitters. For such a system we can actually generate each of the individual signals at passband with the correct center frequency so that we can simply add the resulting individual signals. The individual signals can have different symbol rates, bandwidths and even use different modulation schemes, as long as they are generated at the same sample rate.

## V. Algorithm Implementation

The algorithms developed in this work have been implemented in a software radio system designed and built as part of the SpectrumWare research project at the MIT Lab for Computer Science. This *Virtual Radio* system is a complete, functioning software radio that uses a wideband receiver/down-converter and a specially designed PCI-based I/O system for data transfer. The system can support in excess of 500 megabits per second to and from software applications running in user space on a GPP-based personal computer. Further information on this system can be found in [1], [6], [7].

Current implementations of these algorithms on a 500 MHz Pentium III-based PC can generate a 30kHz narrow-band FM signal at one million samples/second using less than ten percent of the CPU capacity. To perform the interpolation filtering of voice signals and combining of individual narrow-band signals, we use the MMX instruction set to perform multiple operations per instruction.

We have also implemented a digital modulation systems based on the look-up table techniques described above. This system can generate continuous digital modulation signals such as BPSK, CPFSK, PAM or QAM and can, for example, generate an eight-level PAM signal with a symbol rate of 500,000 symbols/second at a output sample rate of 10 MHz.

## VI. Conclusions and Future Work

This work demonstrates how the concepts of DDS can be extended to synthesize digitally modulated waveforms. We have shown how the effect of truncating the transmit filter response can be managed, and we have shown one way to limit table growth as constellation size increases. In this way, these techniques provide an effective way to trade-off memory usage for computational complexity in a digital transmitter. We anticipate that these techniques will become valuable as the memory capacities of systems continue to increase.

Our future work will include efforts to provide more control over table size by generalizing the table mappings to synthesize longer sequences. We also feel that these techniques show promise for implementations which require low-power consumption, due to the inherent advantage of memory devices over computational elements, and will work to further quantify these trade-offs.

## Acknowledgments

This research was supported under the DARPA Global Mobile Communications project and also under a National Science Foundation graduate research fellowship.

## References

[1] Vanu G. Bose and Alok B. Shah. Software Radios for Wireless Networking. In *Proceedings of Infocomm '98.*

[2] Marvin E. Freking. *Digital Signal Processing in Communications Systems.* Van Nostrand Reinhold, 1994.

[3] Leland B. Jackson. *Digital Filters and Signal Processing.* Kluwar Academic Publishers, 1989.

[4] John G. Proakis. *Digital Communications.* McGraw Hill, 1995.

[5] Jeffrey H. Reed. Direct digital synthesis. In *Tutorial notes from PMIRC 1998.*

[6] David L. Tennenhouse and Vanu G. Bose. The SpectrumWare Approach to Wireless Signal Processing. *Wireless Network Journal*, 2(1), 1996.

[7] M. Welborn J. Guttag V. Bose, M. Ismert. Virtual Radios. *JSAC issue on Software Radios*, 1999.