MICROCONTROLLER BASED FUNCTION GENERATOR USING TRIGONOMETRIC DTFS

Tiago Menezes de Lima[1], Lucas Sena Nascimento Lima[1], Brenda Silva de Alencar[1], Jéssica Morais de Andrade[1], Philipe Santana de Almeida[1], Emanuel Benício de Almeida Cajueiro[1]

[1] SENAI CIMATEC, Brazil

Abstract: This paper proposes to showcase the development of an Arduino based function generator utilizing the Discrete Time Fourier Series (DTFS) in its trigonometric form. A low-pass filter coupled with PWM is used as a digital-to-analog converter. The intrinsic relationships when analyzing a signal in the time and in the frequency domains are illustrated. The results focus on consequences to the choices and compromises made, reinforcing their causes. The paper concludes by highlighting the usefulness of the DTFS in its trigonometric form, not so much exposed in the literature if compared to its complex counterpart, and also by suggesting starting points for improving the efficiency and robustness of the obtained results.

Keywords: Arduino; DTFS; function generator; PWM; low-pass filter.

GERADOR DE FUNÇÕES MICROCONTROLADO USANDO SFTD TRIGONOMÉTRICA

Resumo: Este artigo propõe demonstrar a construção de um gerador de funções utilizando a Série de Fourier para Tempo Discreto (SFTD) em sua forma trigonométrica, implementada em Arduino. Um filtro passa-baixas com PWM é usado como conversor digital-analógico. As relações intrínsecas ao analisar o sinal nos domínios do tempo e da frequência são ilustradas. Os resultados focam nas consequências das escolhas e compromissos feitos, reforçando suas causas. O artigo conclui evidenciando a utilidade da forma trigonométrica da SFTD, não tão exposta em literatura se comparada à sua equivalente complexa, e sugerindo ainda pontos de partida para a melhoria da eficiência e robustez dos resultados obtidos.

Palavras-chave: Arduino; SFTD; gerador de funções; PWM; filtro passa-baixas.

1. INTRODUCTION

A function generator is an electronic equipment capable of generating electrical signals with different waveforms and characteristics. Among the numerous ways of constructing these periodic signals, the Discrete Time Fourier Series (DTFS) consists of representing them by means of a weighted sum of sinusoids with different frequencies [1,2].

Even though the DTFS is typically written in a more compact way, in terms of complex exponentials, nothing prevents it from being used in its trigonometric form, either due to hardware restrictions of some microcontrollers or any other reason. The main goal of this paper is to precisely demonstrate this necessity, since the Arduino does not support operations with complex numbers neither natively nor efficiently. Besides, material mentioning the trigonometric form of the Fourier Series in discrete time is scarce in the literature, given that it does not present advantages over the complex form in most engineering fields [2].

The synthesis equations can be seen in (1) and (2):

$$x[n] = a_0 + a_{\frac{N}{2}}(-1)^n + \sum_{k=1}^{\frac{N}{2}-1} a_k \cos\left(k\frac{2\pi}{N}n\right) + \sum_{k=1}^{\frac{N}{2}-1} b_k \sin\left(k\frac{2\pi}{N}n\right), \qquad if\ N\ even \quad (1)$$

$$x[n] = a_0 + \sum_{k=1}^{\frac{N-1}{2}} a_k \cos\left(k\frac{2\pi}{N}n\right) + \sum_{k=1}^{\frac{N-1}{2}} b_k \sin\left(k\frac{2\pi}{N}n\right), \qquad if\ N\ odd \quad (2)$$

, where n refers to the discrete time and N to the period of the function $x[n]$, while the coefficients a_k and b_k may be more easily obtained through its complex counterpart c_k, using the analysis equation in (3) and relations in (4) – it is possible, however, to derive expressions to directly obtain the non-complex coefficients by applying (3) into (4), if needed.

$$c_k = \frac{1}{N}\sum_{k=0}^{N-1} x[n]e^{-jk\frac{2\pi}{N}n} \quad (3)$$

$$a_0 = c_0\,, \qquad a_{\frac{N}{2}} = c_{\frac{N}{2}}\,, \qquad a_k = c_k + c_{-k}\,, \qquad b_k = j(c_k - c_{-k}) \quad (4)$$

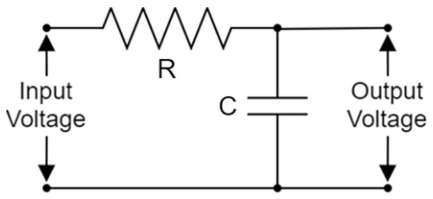
The DTFS not only generalizes the construction of complex signals using elementary functions, moreover, it also allows for a deeper analysis regarding their frequencies. Despite its use in this application having solely demonstrative purposes – since there are much more efficient methods to achieve the effect aforementioned, such as with the use of look-up tables [3] –, this paper's focus is to simply demonstrate the viability of utilizing the DTFS in its trigonometric form through a simple and visual application.

When such signals are required to be presented as analog voltages, a digital-to-analog converter (DAC) is also needed. The Pulse Width Modulation (PWM) technique thus may be used to achieve this effect, leveraging the fact that the average power of a pulse wave is proportional to the ratio of the pulse width to the wave period – this ratio being denominated duty cycle [4,5].

In the frequency domain, a PWM representing a voltage signal ends up having a direct current (DC) component, proportional to its duty cycle, and harmonics that are multiples of its fundamental frequency – i.e., the PWM frequency itself. Therefore, to make use of only the DC component, the signal may go through a filter that attenuates all of its harmonics [5].

Considering the topology of a passive RC low-pass filter, indicated in Figure 1, the resistor R and capacitor C act as a different voltage divider for each frequency, and their values may be adjusted to control at which point the output voltage starts to be significantly attenuated – this point representing the cutoff frequency ω_c. This choice, however, comes with a tradeoff in the filter's response speed, characterized by its time constant τ, as lower cutoff frequencies lead to higher charging times for the capacitor [5,6]. Both of these parameters are depicted in Equation (5).

Figure 1. Passive RC low-pass filter topology



$$\omega_c = \frac{1}{RC} = \frac{1}{\tau} \tag{5}$$

Hence, a PWM with a low-pass filter effectively works as a DAC, where since the duty cycle is proportional to the wave's DC component, one can freely adjust the DC voltage at the filter output.

This paper, therefore, proposes to discuss the project of a function generator based on the DTFS implemented on an Arduino Uno, capable of generating three waveforms (sinusoidal, square wave and triangular wave) and allowing the adjustment of their amplitude (0 to $5\ \text{V}$) and frequency ($0.1\ \text{Hz}$ to $1\ \text{kHz}$).

2. METHODOLOGY

A DAC is first designed by an empirical approach, since it is a simple yet effective method that also avoids directly dealing with the tradeoff between responsiveness and output oscillation. Coefficients for the DTFS are then calculated analytically in order to generate the waves, and lastly a strategy is devised to construct and adjust these signals based on the Arduino Framework.

2.1 Digital-to-Analog Converter

As the Arduino does not have a built-in DAC, one was implemented using the method of a PWM associated with a low-pass filter. In practice, as the filter attenuation is not instantaneous, it is better for the cutoff frequency to be much lower than the first harmonic of the PWM, in order to ensure that only the DC component remains [5,6]. Otherwise, the output voltage may have some noise due to residual harmonics – or, in terms of the time domain, due to a low time constant making the capacitor charge and discharge so quickly that it is not able to maintain a stable voltage at the high/low transitions of the PWM [5]. Similarly, it is of interest that such time constants are also not overly high, or else the filter may take too long to reach its steady state.

On the Arduino, most pins capable of generating a PWM signal do so at around $490\ \mathrm{Hz}$. In order to have more leeway when choosing a cutoff frequency, avoiding considerably low values – and unreasonably high time constants –, the microcontroller's PWM frequency may be adjusted through the appropriate timer registers to approximately $31.4\ \mathrm{kHz}$, maximum possible value on the pin that would be used – some pins allowed for higher frequencies, but at the expense of the proper functioning of Arduino’s time functions [7].

As for designing the filter, while looking for a compromise between system speed and output oscillations, an arbitrary $100\ \mathrm{nF}$ capacitance was fixed and the resistance adjusted until the generated PWM outputted a satisfactorily constant voltage with relatively low time constant, arriving at approximately $500\ \Omega$.

2.2 DTFS Coefficients

Since the generator will only work with periodic signals, just the values in a given period need to be stored, falling upon the algorithm the responsibility to iterate over them repeatedly. These discrete points representing a signal could be calculated in several ways, and the method chosen here involved using the DTFS of each wave – which despite being a rather indirect approach in this scenario, a simple visual application such as this may be a great way to consolidate the applicability of the series in its trigonometric form.

For calculating the DTFS itself, the coefficients were obtained analytically, since this way is more efficient than a numerical approach, computationally speaking. Non-negative functions to be repeated over a period N were arbitrarily chosen to represent the desired waveforms, and Table 1 shows their respective Fourier Series trigonometric coefficients.

Table 1. Waveform base functions and their Fourier coefficients

Waveform	**Function**	**Coefficients**
Sinusoidal	$x[n] = \frac{1}{2} + \frac{1}{2} cos\left(\frac{2\pi}{N} n\right)$	$a_0 = a_1 = \frac{1}{2}$ $a_{\frac{N}{2}} = a_{k,k\neq 1} = b_k = 0$
Square wave	$x[n] = 1\,, \qquad \lvert n \rvert \leq \frac{N-2}{4}$	$a_0 = \frac{1}{2}$ $a_{\frac{N}{2}} = \frac{1}{N} sin\left(\frac{N\pi}{4}\right)$ $a_k = \frac{2}{N} \frac{sin\left(\frac{k\pi}{2}\right)}{sin\left(\frac{k\pi}{N}\right)}$ $b_k = 0$

Triangular wave	$x[n] = 1 - \frac{2\|n\|}{N}, \qquad \|n\| < \frac{N}{2}$	$a_0 = \frac{1}{2}$ $a_{\frac{N}{2}} = \frac{2}{N^2} sin^2\left(\frac{N\pi}{4}\right)$ $a_k = \frac{4}{N^2} \frac{sin^2\left(\frac{k\pi}{2}\right)}{sin^2\left(\frac{k\pi}{N}\right)}$ $b_k = 0$

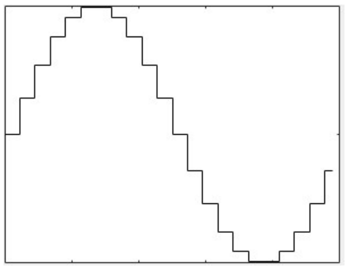
While the coefficients of the sine wave could be obtained merely by inspection of either Equation (1) or (2), those of the square and triangular waves required the application of the analysis equation in (3) and relations in (4), combined with varied algebraic properties [2,8]. It should also be noted that apart from the natural function, the other two were chosen in a constricted manner to ensure that there would be no gaps between the periods of the signal. For that to happen, though, the square and triangular waves require $(N - 2)$ to be divisible by 4 and N to be even, respectively.

2.3 Waves Generation

The DTFS coefficients are first obtained and used to calculate values over a period N, which are stored and then sequentially passed to Arduino's $analogWrite$ function – essentially creating a PWM with duty cycle proportional to the value.

The construction itself of the waves is based on continuously varying the duty cycle in order for the low-pass filter to retain different DC voltages at its output. These, viewed sequentially in time, will visually represent a sine, square or triangular wave, like depicted in Figure 2.

Figure 2. Demonstration of how the sinusoidal signal will be generated



As it can be seen, the frequency of the resulting signal strongly depends on the time of each iteration and the number of points used to completely represent the wave. In this application, therefore, the period N should be low enough to allow frequencies of up to $1\ \mathrm{kHz}$, but still allow for a good enough resolution of the intended signals. Considering all the constraints brought up until now, different values were empirically tested until a N of 22 was chosen, which satisfactorily fulfilled all of the previous conditions – indicating that Equation (1) shall be used instead of (2), when synthesizing the signals.

2.4 Parameters Adjustment

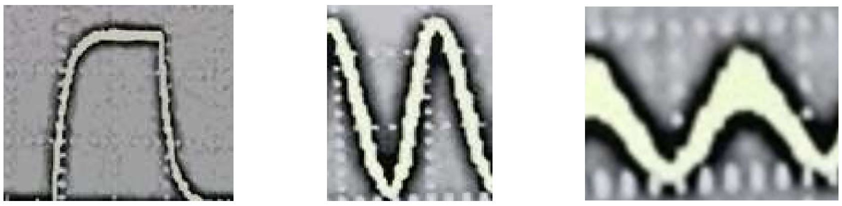
Since the Arduino has a PWM resolution of 8 bits, the amplitude of the waves needs to be remapped into a range of 0 to 255, representing a duty cycle of 0 to 100% and a consequent voltage between 0 and $5\ \mathrm{V}$. Adjusting the frequency, on the other hand, would require a different approach, as directly varying the argument of the function (e.g., $x[2n]$) in discrete time not only modifies its period, but also leads to loss of information by signal compression [2].

As an alternative, since the final wave frequency is directly related to the time of each code iteration, a non-blocking delay with adjustable duration may be used. This logic, however, has an inherent limitation. Even with no delay, the main loop already imposes a minimum duration for the iterations, making the maximum achievable frequency limited by how long or slow the code is. For this application in specific, no delay shall be applied in order to obtain $1\ \mathrm{kHz}$, while for the minimum frequency of $0.1\ \mathrm{Hz}$, an empirical delay of $500\ \mathrm{\mu s}$ came to be required.

3. RESULTS AND DISCUSSION

The methodologies here presented were sufficient to capably generate signals within a wide range of the intended parameters. This section, therefore, focuses on discussing those edge cases where the most eminent limitations were noted.

Figure 3. Limitations observed in an oscilloscope regarding (a) square waves at high frequencies, (b) noticeable harmonics in the sine wave and (c) even higher harmonics at intermediary amplitudes.

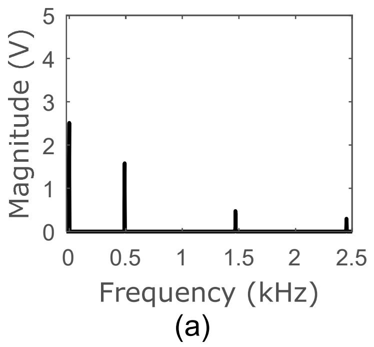

(a) (b) (c)

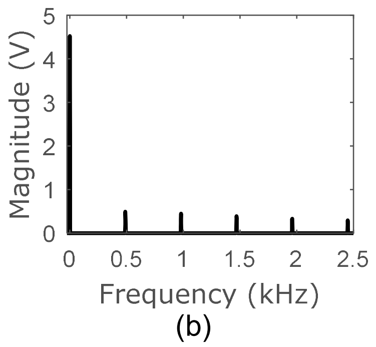
First, Figure 3(a) demonstrates one of the effects of a high time constant on square waves, more noticeable at higher frequencies when the charging of capacitors tends to be comparably slow in relation to the PWM transitions. Increasing this constant even further would risk the signal not even stabilizing before a next call to $analogWrite$ occurs, effectively resulting in data loss.

On the other hand, reducing the time constant would imply increasing the filter cutoff frequency, allowing greater harmonics to pass through. Figure 3(b) illustrates how this would not be a good idea, since the relatively high thickness of the sine wave is due precisely to the harmonics already present in the resulting DC component.

This noisy thickness is even more pronounced at voltages close to $2.5\ \mathrm{V}$, as shown in Figure 3(c). This is due to the fact that to generate these intermediate voltages – based on Arduino's range of 0 to $5\ \mathrm{V}$ –, the PWM needs to have a duty cycle of 50%. When compared to 90%, for example, the former presents higher magnitudes at initial harmonics, as can be seen in Figure 4. This fact, coupled with a noninstantaneous attenuation performed by the non-ideal filter, results in a considerable portion of these harmonics still being present after the filtering.

Figure 4. Frequency Spectra demonstrating the magnitude of harmonics in a PWM at $490\ \mathrm{Hz}$ with (a) 50% and (b) 90% duty cycle.





4. CONCLUSION

This paper proposed a more indirect approach in the construction of waveforms for a function generator, in order to visually showcase the applicability of the DTFS in its trigonometric form – something that is hardly discussed in the literature, in contrast to its complex counterpart, but that may be necessary when the use of complex numbers is not viable.

Furthermore, other secondary accomplished goals involved presenting techniques easily adaptable to other applications, taking the opportunity to also illustrate the intrinsic relationships between the time and frequency domains, as well as some commonly necessary compromises.

Lastly, the analysis brought is also sufficient to further improve the results shown, if desired. Examples include the use of a more selective filtering that would not significantly impact on the time constant; more precise techniques for constructing the waves that do not depend on the code's execution time; or even a higher-frequency-clock microcontroller, allowing for more resolution through larger values of N and providing more room when selecting a filter for the PWM.

Acknowledgments

The authors would like to acknowledge the SENAI CIMATEC University Center for providing the means to execute this project, as well as to thank everyone who contributed with it in any way, especially professors Emanuel Cajueiro, Frederico Ramos, Rodrigo Tutu, and technicians José Gilson Ferreira and Luciano Valverde.

5. REFERENCES

[1] PERES, P. L. D. **Análise de sinais**: Série de Fourier de Sinais Discretos. Campinas, 2015. 50 slides. Available at: <dt.fee.unicamp.br/~peres/ea614/115/pdf/LSS_slides_EA614_Cap4.pdf>. Accessed on: 2 Oct. 2019.

[2] LATHI, B. P. **Sinais e sistemas lineares**. 2ª. ed. Porto Alegre: Bookman, 2007. p. 857.

[3] SALAZAR, A. J. et al. A study on look-up table based sine wave generation. **Proceedings of the Regional Echomail Coordinator**, Porto, Portugal, p. 3-4, 2011.

[4] FLOYD, T. L.; BUCHLA, D. L. Introduction to alternating current and voltage: Nonsinusoidal waveforms. *In*: ______. **Electronics fundamentals**: Circuits, Devices and Applications. 8. ed. rev. London: Pearson, 2014. cap. 8, p. 370-372.

[5] KEIM, R. **Low-pass filter a PWM signal into an analog voltage**. [*S. l.*]: All About Circuits, 11 Apr. 2016. Available at: <allaboutcircuits.com/technical-articles/low-pass-filter-a-pwm-signal-into-an-analog-voltage>. Accessed on: 20 Oct. 2019.

[6] SASAKI, C. M.; OSHIRO, L. K. **Kit didático para implementação de filtros passivos e ativos**. 2014. 145 f. Trabalho de Conclusão de Curso (Curso Superior de Engenharia Industrial Eletrônica), Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2014.

[7] SECRETS of Arduino PWM. **Arduino**, 2019. Available at: <arduino.cc/en/Tutorial/SecretsOfArduinoPWM>. Accessed on: 20 Oct. 2019.

[8] HAYKIN, S.; VEEN, B. V. **Sinais e sistemas**. 1. ed. Porto Alegre: Bookman, 2001. 668 p.