# The Fourier Awakens

## Direct Digital Synthesis

Robbie Katz[†] Benji Lewis[‡] and Cairin Michie[§]
Group 9
EEE4084F Class of 2016
University of Cape Town
South Africa
[†]KTZROB006 [‡]LWSBEN002 [§]MCHCAI001

***Abstract*—This paper concerns the conceptual design of a function generator using the Fourier Series estimation with the use of Direct Digital Synthesis techniques. Various architectures are discussed to determine what is optimal for this design.**

## I. Introduction

Direct Digital Synthesis (DDS) is a method of producing periodic analog signals using digital techniques [1]. In this project, DDS will be used to generate the Fourier Series expansion of a user defined waveform.

DDS allows for periodic signals to be synthesized by constantly referencing a 'template' waveform, which is stored in memory [2]. The frequency of the new waveform is determined by how often values are read from memory. The phase is determined by the point at which reading from memory starts. The analog signal is produced by outputting the digital waveform through a Digital-to-Analog converter (DAC).

DDS was first proposed in 1971 by Tierney, Rader and Gold in the paper published in the IEEE Transactions on Audio and Electroacoustics titled 'A Digital Frequency Synthesizer' [3]. This technique is commonly used in signal synthesis, frequency hopping, medical imaging systems and radio receivers [4]. DDS is beneficial in its ability to generate signals with extreme frequency precision and its ability to change frequency and phase rapidly [5].

The Fourier Series allows for periodic signals to be represented as a sum of sinusoids [6]. Equation 1 [7] shows how this is implemented for an arbitrary waveform $f(t)$, with a period of $2L$.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left( a_n \cos(\frac{\pi n t}{L}) + b_n \sin(\frac{\pi n t}{L}) \right) \tag{1}$$

The individual sinusoids are either cosine or sine, and are characterized by their coefficients, $a_n$ and $b_n$, and their natural frequency. These individual sinusoids can be paired in terms of their frequency, and can both be generated by a DDS module that is sent this frequency as a control word. This represents a single iteration of the summation in equation 1.

This project will be implemented on the Digilent Nexys4™ [8]. FPGA-based acceleration is well suited to this application as it allows any number of DDS modules to be implemented in parallel. The number of DDS modules used will be determined by the number of coefficients used in the summation, which in-turn determines the resolution of the final waveform.

### *A. DDS Overview*

Figure 1 shows the block diagram of a DDS module. The single-cycle 'template' waveform is stored in the Phase Register, and is addressed according to the value in the Phase Accumulator. The address of the next sample to be extracted is determined by the sum of the Phase Accumulator and the contents of the Parallel Delta Phase Register, which contains the tuning word, M.

The tuning word is used to set the frequency of the desired output waveform, $f_o$, and is based on the clock frequency, $f_c$, and the number of bits in the address space, $n$. Equation 2 [9] shows the relationship between M and the output frequency.

$$f_o = \frac{M \cdot f_c}{2^n} \tag{2}$$

The size of the address space, $n$, describes the resolution of the 'template' waveform. On each clock cycle the Phase Accumulator is updated. In the case of $M = 1$, the Phase Accumulator will take $2^n$ clock cycles before it overflows and restarts the cycle. When $M = 2$, the Phase Accumulator will take twice as fast to 'roll over', thus doubling the frequency of the output waveform.

Figure 2 shows the Digital Phase Wheel. There are $2^n$ points on the wheel and on each clock cycle the Phase Accumulator is incremented by M. As shown by the wheel, M defines the jump size around the phase wheel. The greater the jump size, the faster you travel around the wheel, thus resulting in a higher frequency. Consequently, however, the output waveform has a lower resolution for higher values of M. Provided that the template has a high enough resolution, for high values of M, the output will still hold the shape required.

## II. Overview

### *A. Implementation*

Figure 3 provides an overview of the various modules that make up the completed system. An overview of how these modules will be implemented is detailed below.

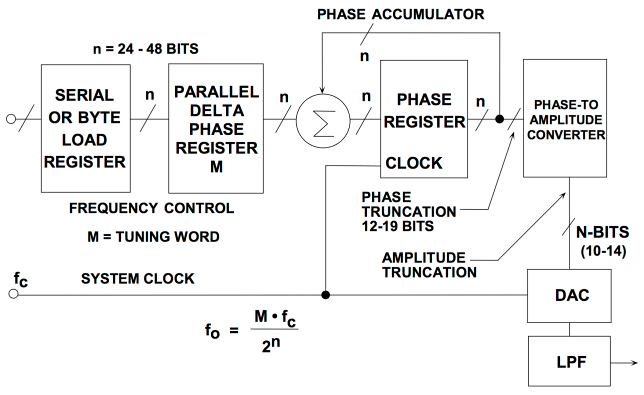
PHASE ACCUMULATOR
n = 24 - 48 BITS
SERIAL OR BYTE LOAD REGISTER
PARALLEL DELTA PHASE REGISTER M
PHASE REGISTER
CLOCK
PHASE-TO AMPLITUDE CONVERTER
n
FREQUENCY CONTROL
M = TUNING WORD
PHASE TRUNCATION 12-19 BITS
AMPLITUDE TRUNCATION
N-BITS (10-14)
$f_C$
SYSTEM CLOCK
DAC
LPF
$f_O = \frac{M \cdot f_C}{2^n}$

Fig. 1. DDS Block Diagram [9]

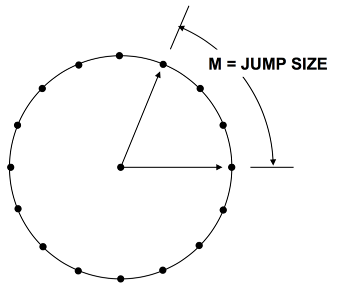

Fig. 2. Digital Phase Wheel [9]

The user defines a waveform they wish to be synthesized on the PC based Graphic User Interface (GUI). The user has control over certain parameters of the waveform. Namely, the type of waveform, amplitude, frequency, and the number of Fourier coefficients used in generating the output. These parameters will need to be within reasonable constraints, as restricted by the implementation. These constraints will be enforced through the GUI.

The Fourier coefficients and frequencies corresponding to the users waveform will be calculated in Matlab. The parameters along with the Fourier coefficients will then be passed to the FPGA via the on board UART port as serial data.

A high resolution sine wave will be stored in ROM. The wave will have a resolution of 32 bits for both the amplitude and period. This will allow for the Output waveform to have a maximum amplitude and period of $2^{32}$ data points. This allows for an accurate output waveform through a larger range of frequencies. The DDS modules will step through the stored sine wave according to the tuning word, M. As M is increased, the synthesized waveform's frequency increases.

The DDS modules will shift the sine wave by $\frac{\pi}{2}$ to form a cosine when this is required in the Fourier calculation. Each DDS module will synthesize a single pair of sine and cosine functions based on their frequency by extracting the point from the look-up table, after which these waveforms will be amplified according to their corresponding coefficients. The outputs of the individual DDS modules will then be summed to produce final output waveform. The synthesized waveform will then be output over the audio jack using Pulse Width Modulation (PWM).

The design will implement 45000 DDS modules. This number was decided on so that the output would safely span through the human audible frequency range of 20Hz to 20kHz [10], and satisfy the Nyquist sampling criterion. The 45000 DDS modules give sufficient bandwidth for the human audible range but is not restricted to any specific frequency range.

### B. Interface

The user will interact with the FPGA through the custom built GUI. The interface will allow the user to specify the following properties of the output waveform:

1) Amplitude
2) Frequency
3) Number of Fourier coefficients used
4) Waveform shape

The synthesized waveform will be output to the audio output, and can be viewed on an oscilloscope, or used as a function generator.

## III. Detailed Design

In this section, detailed descriptions of each of the modules will be elaborated upon.

### A. UART

To receive commands from a PC, UART communication is implemented. Only the UART receive needs to be implemented as the design only requires for commands to be received and will not ever have to send any data. The UART communication protocol is used to send data specifying the Fourier coefficients and required tuning words for each of the DDS modules. The baud rate of the UART communication effects how fast the design can change between waveforms. With consideration to this fact, a 100Mhz baud rate is implemented. The protocol implemented uses the standard UART protocol for receiving data. A UART packet has 10 bits. A start bit, 8 bits of data and an end bit. The start bit is indicated by the line going from high to low. The end bit is indicated by the line going high.

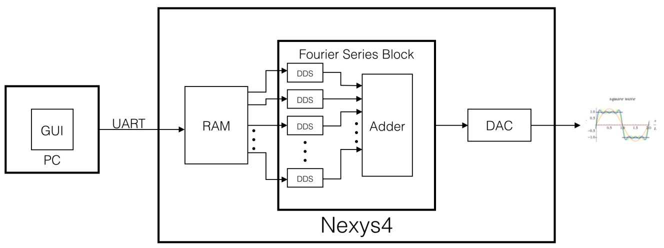

Fig. 3. General System Block Diagram

## B. DDS

The classic DDS implementation will be used with a slight twist. The actual DDS module will be implemented using the traditional look up table method, however each DDS module will produce two waveforms simultaneously. A sine and cosine waveform will be produced with the same frequency but different amplitudes in each DDS module. As shown in equation 1, when approximating a waveform, a sine and cosine waveform of each natural frequency is needed. To be more efficient with the available resources it is logical to generate both the sine and cosine waveform in each DDS module. This is done for two reasons, firstly the waveforms need to be generated at the same frequency and secondly the ROM blocks can be dual access, meaning the number of look up tables needed can be limited.

With consideration to the generation of the two wave forms, there were two options available. The first was to use a look-up table. This is a very fast method which requires only one fetch cycle to retrieve values, however it requires vast amounts of spacial resources to implement. The second option was to calculate waveform values on the fly. This uses no spacial resources, however, this requires more clock cycles to calculate. To calculate the waveform values on the fly the calculations laid out in equation 3 [11] would have to be implemented.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots \tag{3}$$

There are techniques that can be used to make the calculation faster, such as factoring the equation to ensure it is predominantly an unsigned summation. But using this technique will always limit performance. Specifically the number of clock cycles required to calculate a waveform value will limit the maximum frequency that can be output.

Considering all the details specified above, the final DDS module was designed as shown in Figure 4.

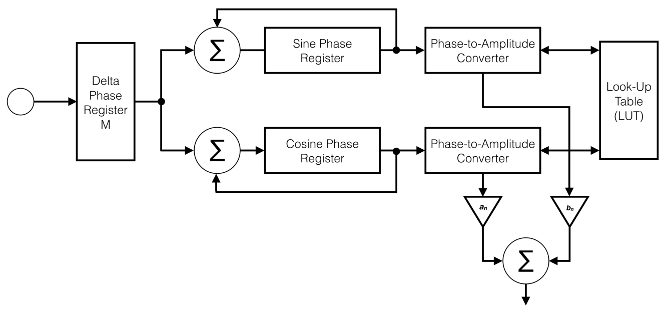

Fig. 4. DDS module

The module receives three values, the tuning word M and the amplitude words $a_n$ and $b_n$. The tuning word specifies the frequency of the sine and cosine waves generated as shown in equation 2.

By defining a clock speed and a period length of $2^n$ bits long, the smallest frequency that can be output is defined. By increasing M, frequency is increased. The amplitude words $a_n$ and $b_n$ define how the sine and cosine amplitude values are scaled, these values correspond to those specified in equation 1. The sine phase register stores the current phase point that needs to be output. The cosine phase register acts identically to the sine phase register but has a $\frac{\pi}{2}$ phase shift. The phase to amplitude converters check the look-up table for the corresponding value, and output a scaled version corresponding to the coefficients. Finally, the two values are added together, to be used in producing the final output waveform.

## C. ROM

For the design it was decided that using ROM instead of RAM would be sufficient. This is because the values in the look-up table are never changed. The ROM, as discussed for the DDS module, needs to be dual access. This is so that a DDS module can output a sine and cosine wave simultaneously. The waveform stored in the look-up table is a sine wave. To use the look-up table for cosine waves, a $\frac{\pi}{2}$ shift must be applied. The number of values stored in the ROM is $2^{32}$ with each value having 32 bits. This means that the amplitude resolution of the waveform is 32 bits and the resolution of the period is also 32 bits.

## D. Adder and PWM

The Adder takes all the output values from the DDS modules, adds them together and outputs the final value. Finally, the value on the output of the Adder is passed into the PWM module. The PWM module interprets this value as a duty cycle. The duty cycle is defined by equation 4:

$$DutyCycle = \frac{AdderOutput}{2^{32}} \tag{4}$$

The PWM module outputs a PWM signal with the calculated duty cycle through a filter that converts PWM to an analogue output.

## IV. Architecture Implementation

The performance of this design is vastly affected by the architecture it is implemented on. It is important to investigate the various possible architectures available to determine which will have the best performance.

## A. Single Core Implementation

Using a single CPU would be the worst case scenario. Various aspects of our design will have very poor performance on a single CPU. The first module that will have poor performance, due to a serial implementation, is the DDS module. The complexity of the performance, as the number of DDS modules grows, is linear - $O(n)$. For each module added there is a constant increase in computation time. With $45000$ modules this would create a delay that would inhibit the system, preventing real time synthesis.

Furthermore, the adder stage of the design will also have poor performance, specifically a linear increase with more

DDS modules. Each added module would require an additional summation instruction resulting in a $O(n)$ complexity.

Based on these downfalls, using a single CPU is not scalable. It may work for small numbers of DDS modules, but as the problem scales up, the CPU will fall behind.

### B. Multiple-Core Implementation

A multi-core implementation would be a great improvement compared to using a single CPU. The DDS module part of the proposed design is embarrassingly parallel. This means that an increase in threads would result in a direct increase in speed-up, assuming each thread is running on a different processor. This means that with sufficient processors, the DDS modules would have constant complexity - $O(1)$. The addition stage of the design will not, however, have the same performance. With each addition only two numbers can be added together at a time. This means after each addition step, the number of further additions to make halves. The complexity is therefore $O(log(n))$. This addition complexity is also dependent on there being enough processors to accommodate all the additions.

The multi-core implementation has a very good theoretical performance, however it is dependent on having a huge number of processors. For the proposed design, it would be required to have tens of thousands of processors. If implemented on an architecture with too few processors, the design will result in poor performance, with the risk of not being able to process in real time. Seeing as an architecture with tens of thousands of processors is not common and would cost millions of dollars, this is not a feasible design.

### C. GPU Implementation

Due to the parallel nature of the design proposed, a GPU solution seems to be a very logical route to take. Using a GPU is a much more suitable architecture compared to the multi-threaded implementation due to the Single Instruction Multiple Data (SIMD) [12] nature of the design. This means that each parallel cell in the GPU will have the same performance as a thread on a different processor in the multi-threaded implementation. This means that a using a GPU will have the same performance as the multi-threaded implementation, but will be more feasible as GPUs are made with many parallel units that are relatively cheap. Despite the density of parallel units in a GPU, there are seldom GPUs that have tens of thousands of parallel units. This means that it will still be very expensive to implement this design on GPUs.

### D. FPGA Implementation

The final possible implementation is on an FPGA. With an FPGA, each module can be implemented separately and will run concurrently. This means that the complexity for the DDS modules will be $O(1)$. The complexity of the adder will be $O(log(n))$ as seen in the multi-core and GPU implementations. However, on the FPGA the additions will not be performed on clock cycles. This means that the addition will be implemented as a circuit, so while the complexity is $O(log(n))$, the time delay grows very slowly with increasing number of modules, because the complexity refers to the growth of the gate delay rather than clock cycles. The FPGA is the best architecture for this design, particularly due to its modular and parallel nature. This is the architecture that the proposed design will be implemented on.

## V. Conclusion

In conclusion, this project focuses on generating the Fourier Series using parallel DDS modules on the Digilent Nexys4™ [8]. Users will be able to enter the desired waveform on a PC based GUI. The Fourier Series will then be computed on the FPGA based on the coefficients and frequencies received via UART. The final waveform is then outputted through the on-board Audio port using PWM.

FPGA-based acceleration is well suited to this application as this architecture is optimal when compared to CPU's or GPU's. A distributed memory-model will be implemented for each DDS module in the form of ROM.

The system is currently in the prototyping phase, and tests will be run to determine the overall performance achieved. The results obtained will be compared to a benchmark that will be executed on a standard PC.

In the final paper, the completed design will be detailed and the performance results will be discussed. From this, conclusions will be drawn and future work will be outlined.

## References

[1] M. Christiano, "Everything You Need to Know About Direct Digital Synthesis," http://www.allaboutcircuits.com/technical-articles/direct-digital-synthesis/, November 2015.

[2] N. Instruments, "DDS Waveform Generation Reference Design for LabVIEW FPGA," http://www.ni.com/example/31066/en/, January 2016.

[3] B. G. Joseph Tierney, Charles M. Rader, "A Digital Frequency Synthesizer," *IEEE Transactions on Audio and Electroacoustics*, vol. 19, no. 1, pp. 48 – 57, March 1971.

[4] Lancaster Hunt, "Direct Digital Synthesis (DDS) For Idiots ( like me )," http://lancasterhunt.co.uk/direct-digital-synthesis-dds-for-idiots-like-me/, November 2009.

[5] N. Instruments, "Understanding Direct Digital Synthesis (DDS)," http://www.ni.com/white-paper/5516/en/, May 2015.

[6] E. W. Weisstein, "Fourier Series," http://mathworld.wolfram.com/FourierSeries.html.

[7] "Fourier Series," https://www3.nd.edu/nancy/Math30650/Matlab/Demos/fourier_series/fourier_series.html.

[8] Digilent, "Nexys4 FPGA Board Reference Manual," http://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPNexys4/documenatation/Nexys4_RM_VB1_Final_3.pdf, September 2013.

[9] A. Devices, "Fundamentals of Direct Digital Synthesis (DDS)," http://www.analog.com/media/en/training-seminars/tutorials/MT-085.pdf, 2009.

[10] S. W. Smith, "Human Hearing," http://www.dspguide.com/ch22/1.htm.

[11] Dotan Cohen, "Taylor series for sine or sinus," http://dotancohen.com/eng/taylor-sine.php/.

[12] "GPU Parallelizable Methods," http://www.oxford-man.ox.ac.uk/gpuss/simd.html.