Realization and Testing of Transfer Functions on a FPGA Platform

A REPORT on B. Tech Project Part-2

(EE4291)

BY

MURLI MANOHAR MISHRA: 2020EEB041
AYUSH SAGAR: 2020EEB045
ADARSH JAISWAL: 2020EEB050
SOUGAT MAHATO: 2020EEB076
BIPRO BHADRA: 2020EEB097

UNDER THE GUIDANCE OF

Prof. Debjani Ganguly

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

2023-2024

DECLARATION BY THE CANDIDATES

I certify that to the best of my knowledge

i) The work contained in the thesis has been done by myself/ourselves under the general supervision of my/our supervisor/supervisors.

ii) I/We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

iii) Whenever I/we have used materials (data, theoretical analysis and text) from other sources, I/we have given due credit to them by citing them in the text of the report and giving their details in the references.

iv) Whenever I/we have quoted written materials from other sources, I/we have put them under quotation marks and given due credit to the sources by citing them and giving details in the references.

SL No.	Name	Examination Roll No.	Signature
1	Murli Manohar Mishra	2020EEB041	
2	Ayush Sagar	2020EEB045	
3	Adarsh Jaiswal	2020EEB050	
4	Sougat Mahato	2020EEB076	
5	Bipro Bhadra	2020EEB097	

Date:

Place:

CERTIFICATE

I, do hereby, forward the report on B. Tech. Project Part-2 entitled "**Realization and Testing of Transfer Functions on a FPGA Platform"** prepared by **Murli Manohar Mishra, Ayush Sagar, Adarsh Jaiswal, Sougat Mahato and Bipro Bhadra** under my guidance, as partial fulfillment of requirements for the completion of degree of B.Tech (Electrical) at IIEST, Shibpur.

_____

Debjani Ganguly
Associate Professor
Department of Electrical Engineering
IIEST, Shibpur

_____

Dr. Anindita Sengupta
Professor and Head
Department of Electrical Engineering
IIEST, Shibpur

CERTIFICATE OF APPROVAL

We recommend that the report on B. Tech Project Part-1 entitled "**Realization and Testing of Transfer Functions on a FPGA Platform**" submitted by **Murli Manohar Mishra (2020EEB041), Ayush Sagar (2020EEB045), Adarsh Jaiswal (2020EEB050), Sougat Mahato (2020EEB076) and Bipro Bhadra(2020EEB097)** to the Department of Electrical Engineering, Indian Institute of Engineering Science and Technology, Shibpur be accepted in partial fulfillment of the requirements for the degree of **Bachelor of Technology** in **Electrical Engineering** from this institution after successful defence of the work by the candidates in the viva-voce examination held on 10.05.2024.

Board of Examiners

1. _____ Signature with date

   _____ Name of the examiner

2. _____ Signature with date

   _____ Name of the examiner

3. _____ Signature with date

   _____ Name of the examiner

4. _____ Signature with date

   _____ Name of the examiner

5. _____ Signature with date

   _____ Name of the examiner

ACKNOWLEDGEMENT

First and foremost, the team would like to express its sincerest gratitude to our supervisor Prof. Debjani Ganguly for her continued support and guidance on every step of the way. Through her immense experience and knowledge, she not only helped us to overcome the challenges faced during this project but also played a vital role in our learning.

The team would also like to express its thanks to the Department of Electrical Engineering, IIEST, Shibpur for the opportunity to work on this project and for providing the necessary equipment thereby enabling us to gain practical experience on some of the concepts taught to us during the semester.

Lastly, the team extends its gratitude to everyone who was involved in this project directly or indirectly. Each contribution, no matter how small, has played a significant role in helping this project reach this stage. Thank you to everyone who has been a part of this enriching journey so far.

Date: _____

Murli Manohar Mishra
(2020EEB041)
Ayush Sagar
(2020EEB045)
Adarsh Jaiswal
(2020EEB050)
Sougat Mahato
(2020EEB076)
Bipro Bhadra
(2020EEB097)

Contents

Table of Figures

List of Tables

1. Introduction

The transfer function of an LTI system, *G(s)*, is defined as the ratio of the Laplace transform of the input (*C(s)*) and the Laplace transform of the output (*R(s)*), with all initial conditions set to zero.

$$G(s) = \frac{C(s)}{R(s)}$$

Where *C(s)* and *R(s)* may be considered as two polynomials (of *s*). The transfer function defines the dynamic behaviour of the system in the *s* domain. So, the transfer function of a Single Input Single Output (SISO) system can be considered to represent its behaviour truthfully. If the transfer function of a SISO controller is thus realized, the controller may be assumed to have been built. Such transfer functions/controllers can be implemented using discrete hardware in the form of op-amps. But these circuits suffer from the drawbacks of:

- Large size for transfer functions of higher order
- Inflexibility - once built, even minimal changes will be hard to implement.

Digital controllers, on the other hand, are free from the above drawbacks. However, they can implement the transfer functions only in the discrete domain.

Generally, the digital controllers bring advantages like heightened precision, flexibility, and programmability, proving essential in tasks such as power electronics, where they efficiently regulate electrical power conversion, optimizing energy use. Moreover, in communication systems, these controllers enable accurate signal processing.

Serving as the cognitive core of embedded systems, these controllers furnish the intelligence and control needed for various applications. Examples range from commonplace microcontrollers in household devices to industrial automation's programmable logic controllers (PLCs). In the automotive sector, digital controllers manage engine performance, optimize fuel efficiency, and facilitate advanced driver assistance systems. The adaptability of digital controllers across diverse tasks, combined with their adeptness at processing complex algorithms, underscores their critical role in advancing Electrical Engineering capabilities and driving technological innovation.

Realization of the digital controller can be done using a microcontroller or a DSP or an FPGA. The FPGA provides a parallel computing platform so that fast processing of the input variables is possible and the output can be provided in real time. In the earlier semester, realization of the controller transfer function in the FPGA was made possible. However, to have a look at the output, a DAC interface was necessary so that the output (corresponding to, say, a given step input) can be observed on the oscilloscope. An I2C DAC was programmed to serve this purpose. However, for taking the inputs to the controller, an ADC is required. The inputs can include a reference value and a feedback value from a sensor (voltage/current/temperature).

This report describes the design and implementation of a digital controller on Altera Cyclone II EP2C5T144I8N/Cyclone IV EP4CE115F29C7N FPGA. Verilog HDL has been used for the design and implementation of the controllers. The programming and simulations have been done in Quartus II 13.0 and ModelSim-Altera 10.1d.

The report discusses the methodologies used for the implementation of the digital controller, simulations performed to verify the results and the hardware used, providing a detailed account of our work.

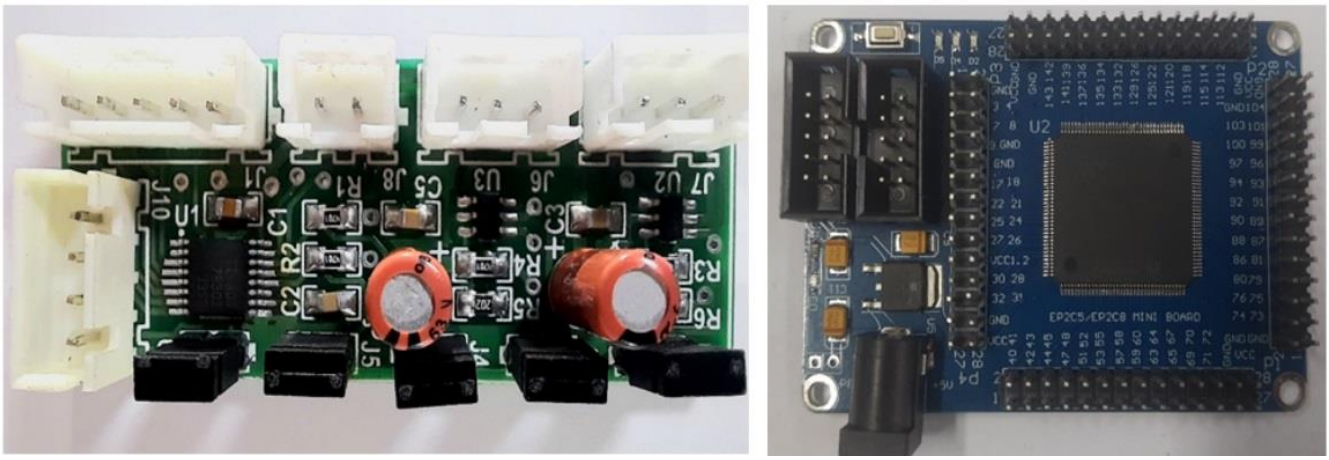


Figure 1: ADC - DAC Module (left) & Altera Cyclone II FPGA (right)

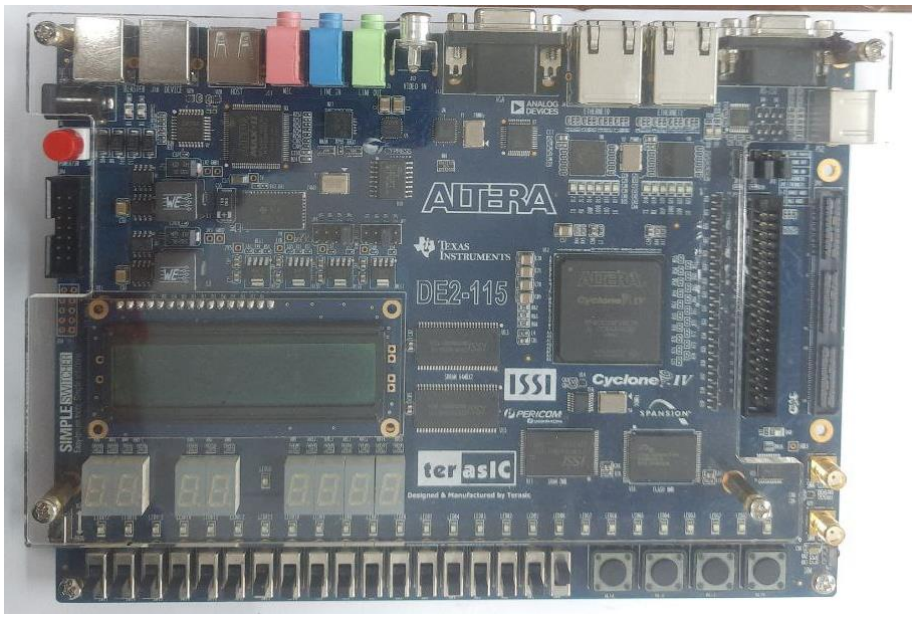


Figure 2 Altera Cyclone IV FPGA

2. Literature Review

The concept of this work was inspired by a paper on the development of a control systems laboratory experiment. [1] This paper showed a novel way of emulating a transfer function using a low-cost microcontroller in real time.

The “Implementation of an IIR filter on FPGA from scratch” by Pablo Trujillo [2] describes in great details how to get to the difference equation from a transfer function in the Laplace domain to z-domain and then its implementation in Verilog HDL. In the article, the author has used bilinear transformation stating its ease of use and implementation to convert from Laplace domain to the z-domain. The author has also showed how the real numbers which are present as the coefficients in the transfer function can be handled using fixed point representation in Verilog. The IIR filter needs its past values of input as well as the output as indicated by its transfer function. To achieve this the author has used a pipeline structure to store the past values.

An important thing to note is that in the article, the author has pointed out that bilinear transformation introduces an error known as frequency warping.

“In case of the bilinear transform, one of the errors that we will introduce is the *frequency warping*. This effect produces a shift of the natural frequency of the filter.” [2]

The development of the Verilog code for the implementation of the transfer function requires an understanding of the different assignments used in Verilog, namely the blocking and nonblocking assignment. The paper “Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill!” by Clifford E. Cummings [3] describes the scheduling of the blocking and nonblocking assignments and suggests some important guidelines so that correct circuit could be synthesized. The paper contains a section about “Pipeline Modelling” and the correct way to synthesize it using Verilog. The pipeline model has been used in the implementation of the transfer function in Verilog as the difference equation requires previous state values of the input and output.

3. Digital controller Implementation

3.1 Transfer Function:

"The *transfer function* of a linear, time-invariant, differential equation system is defined as the ratio of the Laplace transform of the output (response function) to the Laplace transform of the input (driving function) under the assumption that all initial conditions are zero." [3]
The behavior of the digital controller can be described using the transfer function represented in the z-domain. To achieve this, we first describe the transfer function in the Laplace domain for the controller and then transform it to the z-domain using either the Forward Euler Method or the Backward Euler Method or the Tustin Method (also known as bilinear transformation).

3.1.1 Discretization of Continuous Domain Transfer Function

The continuous domain transfer function can be converted to the discrete domain by using one of the methods below, with each method having its own pros and cons:

a. Forward Euler Discretization:

This method is also known as difference method, which approximates the time derivative with a forward difference. It is the simplest method to implement but least accurate than other methods. In forward Euler method the variable s is replaced by:

$$s = \frac{z-1}{T_s} \tag{3.1}$$

where T_s = sampling time in second

b. Backward Euler Discretization:

Backward difference method has improved accuracy over Forward Euler method, but it adds more complexity to the computational hardware. In backward Euler method the variable s is replaced by:

$$s = \frac{z-1}{zT_s} = \frac{1-z^{-1}}{T_s} \tag{3.2}$$

where T_s = sampling time in second

c. Trapezoidal or Tustin Discretization:

Tustin method, having better accuracy than Forward Euler method and lesser complexity than Backward Euler method, is more preferable. In Tustin method the variable s is replaced by:

$$s = \frac{2}{T_s}.\frac{z-1}{z+1} \tag{3.3}$$

where T_s = sampling time in second

The chosen transfer function for implementation on FPGA is G(s), which can be modified as per the requirements.

$$G(s) = \frac{-2.835\text{x}10^{-19}s^3 + 1.905\text{x}10^{-13}s^2 - 6.629\text{x}10^{-9}s + 6.4\text{x}10^{13}}{s^3 + 9.997\text{x}10^4 s^2 + 4.8\text{x}10^9 s + 6.528\text{x}10^{13}} \quad (3.4)$$

"Astrom and Wittenmark (1984) have developed a guideline for selecting the sampling interval, Ts. Their conclusion is that the value of T_s in seconds should be in the range of $0.15/w_m$ and $0.5/w_m$, where w_m is the zero dB frequency (rad/s) of the magnitude frequency response curve for the cascaded analog compensator and plant." [4]

So, for the given transfer function, this range of T_s comes to (7ms – 25ms). Using a sampling time of 10ms (sampling frequency of 100 kHz) and applying Tustin's transformation to the transfer function, we obtain the z-domain transfer function:

$$G(z) = \frac{0.004914 + 0.01474 * z^{-1} + 0.01474 * z^{-2} + 0.004914 * z^{-3}}{1 - 2.061 * z^{-1} + 1.477 * z^{-2} - 0.3759 * z^{-3}} \quad (3.5)$$

3.2 Implementation of Transfer Function in Verilog

Implementing a discrete transfer function in Verilog involves representing the transfer function in a discrete-time form and using Verilog constructs to calculate the output based on the input and past outputs. The specific implementation depends on the order of the transfer function and whether it is direct form or state-space form.

For a direct form implementation, the transfer function is expressed as a difference equation, which relates the current output (y[n]) to past inputs (x[n-k]) and past outputs (y[n-m]).

$$y[n] = b_0*x[n] + b_1*x[n-1] + ... + b_p*x[n-p] - a_1*y[n-1] - ... - a_q*y[n-q] \quad (3.6)$$

where:

- y[n] is the discrete output of the controller
- x[n] is the discrete input to the controller
- b_0, b_1, ..., b_p are the coefficients of the numerator of the transfer function
- a_1, a_2, ..., a_q are the coefficients of the denominator of the transfer function
- p is the order of the numerator of the transfer function
- q is the order of the denominator of the transfer function

The derived z-domain transfer function can be converted to a difference equation by taking inverse z-transform of it.

$$y[n] = 0.004914 * x[n] + 0.01474 * x[n-1] + 0.01474 * x[n-2] + 0.004914 * x[n-3] + 2.061 * y[n-1] - 1.477 * y[n-2] + 0.3759 * y[n-3] \quad (3.7)$$

The difference equation requires that the past values of the input as well as the output be stored for calculation of the next stage result. To achieve this, a pipeline structure is used to capture and store the past values. The values get shifted in this pipeline structure of every clock pulse and stored in a variable.

3.3 Real number representation

In the digital systems, real numbers are represented using various numerical formats, each with its own advantages and disadvantages. The two types of representations are floating-point numbers and fixed-point numbers.

Floating-Point Numbers

Floating point number have two components: Mantissa and Exponent, just like the scientific notation. These numbers can be single precision (32-bit) or double precision (64-bit), and can represent a wide range of values, however, working with them requires high computing power. This is why floating-point units (FPUs) exist solely to perform calculations with the floating-point numbers. Therefore, with this representation of real numbers, we get very good accuracy, but the cost of computing increases as additional FPUs are required.

Fixed-Point Numbers

Fixed-point numbers are another way of representing real numbers. These make use of binary scaling to represent real numbers in the form of integers. Thus, the range of numbers that we can represent is fixed but the advantage that we get is that we can have integer like operations which is fast and does not require any additional computing resources like the FPUs.

The fixed-point numbers can be represented using Q-format. In Q format, we specify the number of integer bits and the number of fractional bits as: Qm.n where m denotes the no. of integer bits and n denotes the no. of fractional bits. For a Qm.n number, the range is from (-2^m) to $(2^m - 2^{-n})$.

To deal with the real numbers in the coefficients of the transfer function of the controller, the fixed-point representation has been chosen because of the fact that the controller will be implemented on a low-end FPGA. Using fixed-point representation will ensure that the controller can function without the need for an external FPU unit.

The coefficients of the difference equation are of range of 10^-3, so to represent these numbers, Q3.12 has been chosen.

For example, b0 = 0.008496 can be represented in Q3.12 as 22h.

4. Response Analysis of Transfer Function

4.1 Time Response

Three diverse transfer functions have been selected to evaluate the controller's performance and its consistency across various transfer function orders and time response requirements. Among these transfer functions are those derived from the Boost converter model, such as the output voltage to duty ratio and output voltage to input voltage relationships.

Transfer function 1:

$$G_1(s) = \frac{-2.835\text{x}10^{-19}s^3 + 1.905\text{x}10^{-13}s^2 - 6.629\text{x}10^{-9}s + 6.4\text{x}10^{13}}{s^3 + 9.997\text{x}10^{4}s^2 + 4.8\text{x}10^{9}s + 6.528\text{x}10^{13}} \quad (4.1)$$

The step response of the continuous domain transfer function,

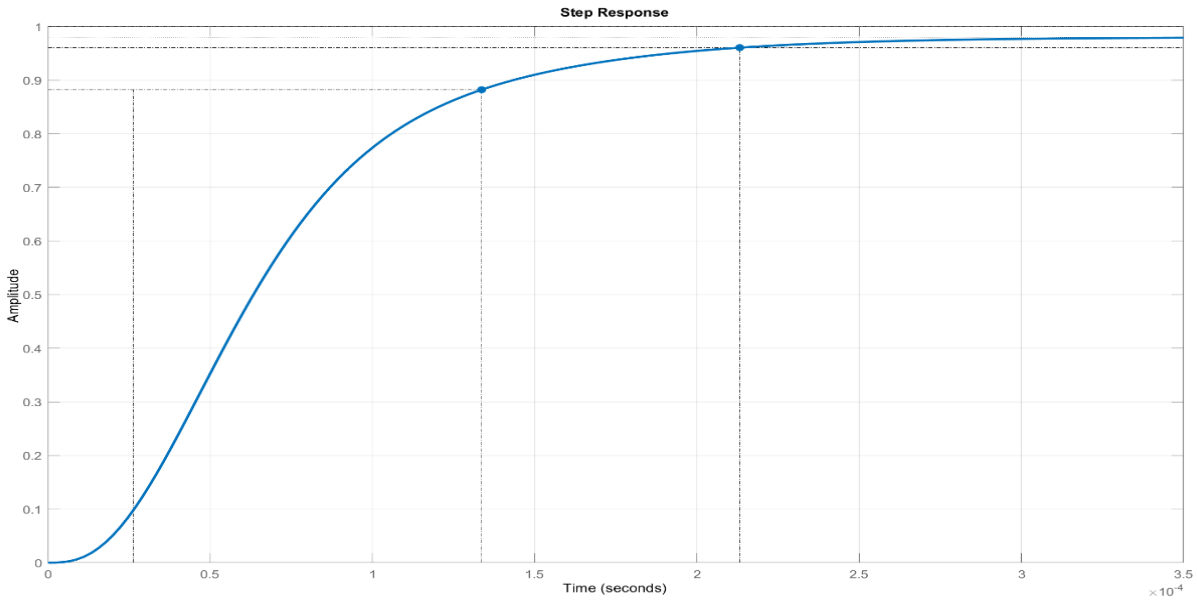


Figure 3: Continuous domain step response of the system in Matlab

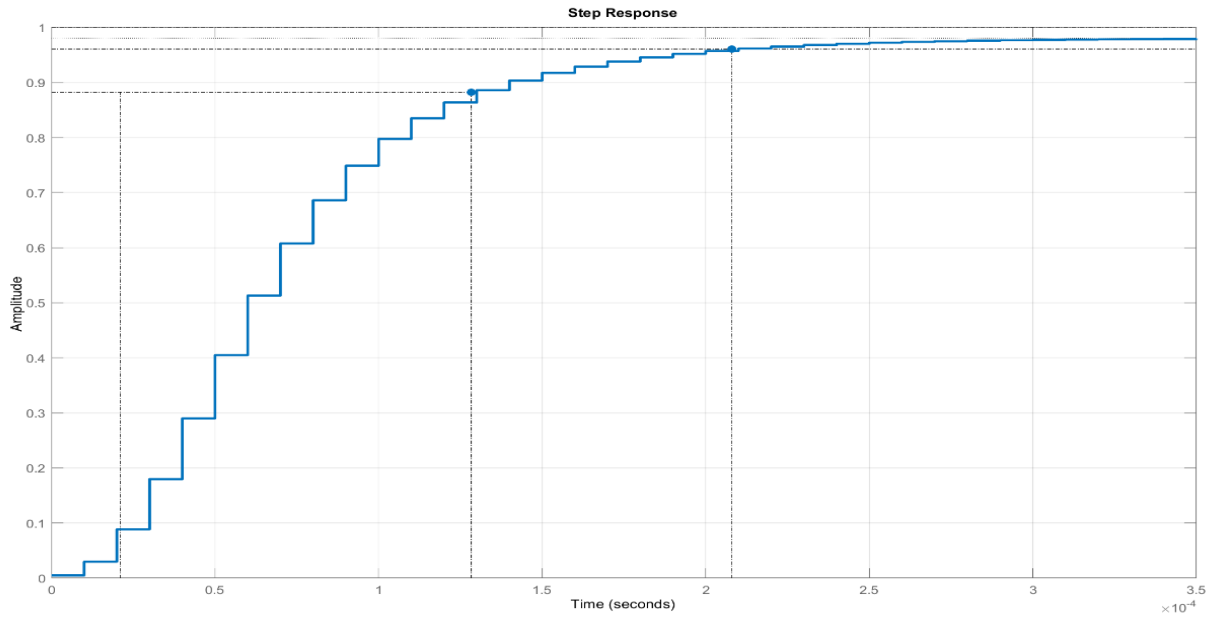


Figure 4: Discrete domain step response of the system in Matlab

The step response of the discrete domain transfer function,

$$G_1(z) = \frac{0.004914+0.01474*z^{-1}+ 0.01474*z^{-2}+0.004914*z^{-3}}{1- 2.061*z^{-1}+1.477*z^{-2} -0.3759*z^{-3}} \quad (4.2)$$

And the difference equation corresponding to the above z-domain transfer function is,

$$Y(z)(1 - 2.061 * z^{-1} + 1.477 * z^{-2} - 0.3759 * z^{-3}) = X(z)(0.004914 + 0.01474 * z^{-1} + 0.01474 * z^{-2} + 0.004914 * z^{-3}) \quad (4.3)$$

Solving it further,

$$Y(z) = 0.004914 * \mathrm{X(z)} + 0.01474 * z^{-1}\mathrm{X(z)} + 0.01474 * z^{-2}\mathrm{X(z)} + 0.004914 * z^{-3}\,\mathrm{X(z)} + 2.061 * z^{-1}\mathrm{Y(z)} - 1.477 * z^{-2}\mathrm{Y(z)} + 0.3759 * z^{-3}\,\mathrm{Y(z)} \quad (4.4)$$

Taking z-inverse of the above equation,

$$\mathrm{y[n]} = 0.004914 * \mathrm{x[n]} + 0.01474 * \mathrm{x[n-1]} + 0.01474 * \mathrm{x[n-2]} + 0.004914 * \mathrm{x[n-3]} + 2.061 * \mathrm{y[n-1]} - 1.477 * \mathrm{y[n-2]} + 0.3759 * \mathrm{y[n-3]} \quad (4.5)$$

The code for the controller in Verilog has been attached in Appendix A.

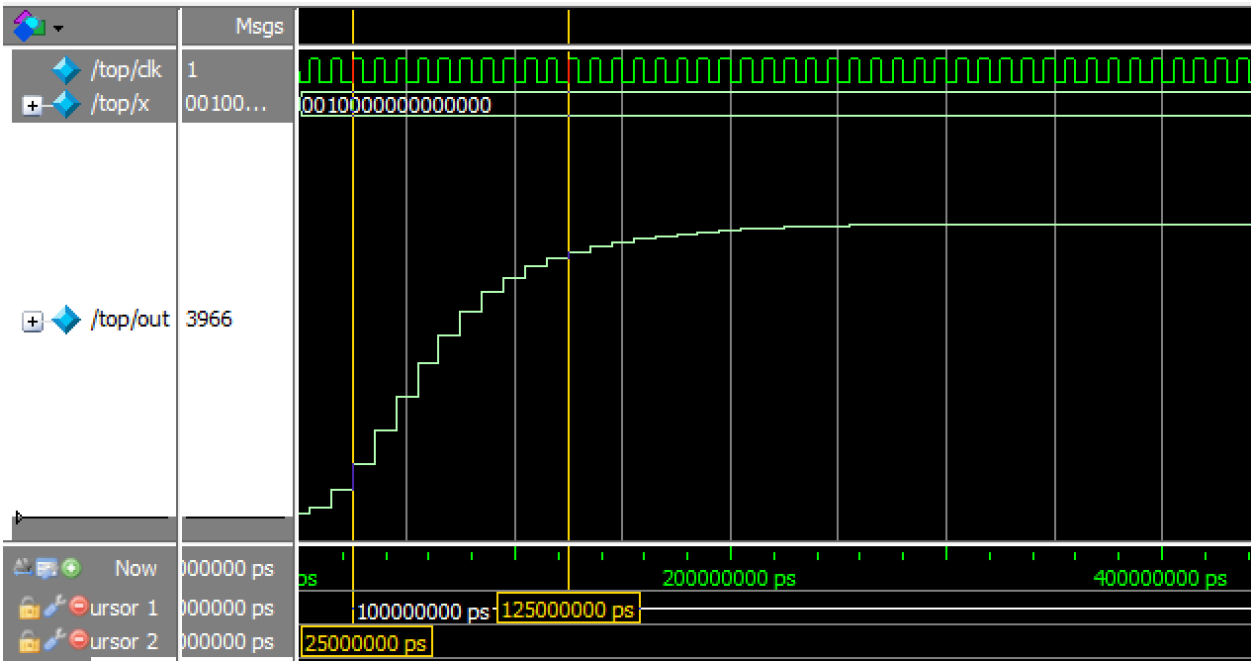


Figure 5: Step Response of TF in Verilog indicating Rise Time

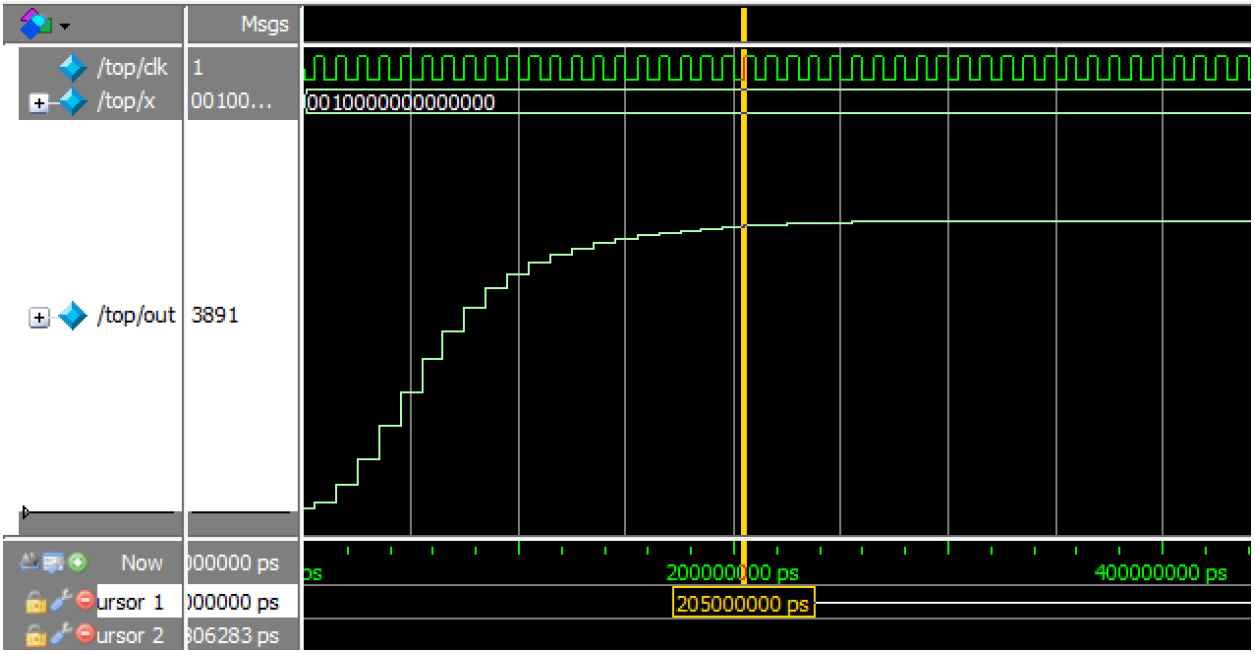


Figure 6: Step Response of TF in Verilog indicating Settling Time

Transfer function 2:

$$G_2(s) = \frac{0.833}{5.798\text{x}10^{-7}s^2 + 5.807\text{x}10^{-4}s + 0.6939} \tag{4.6}$$

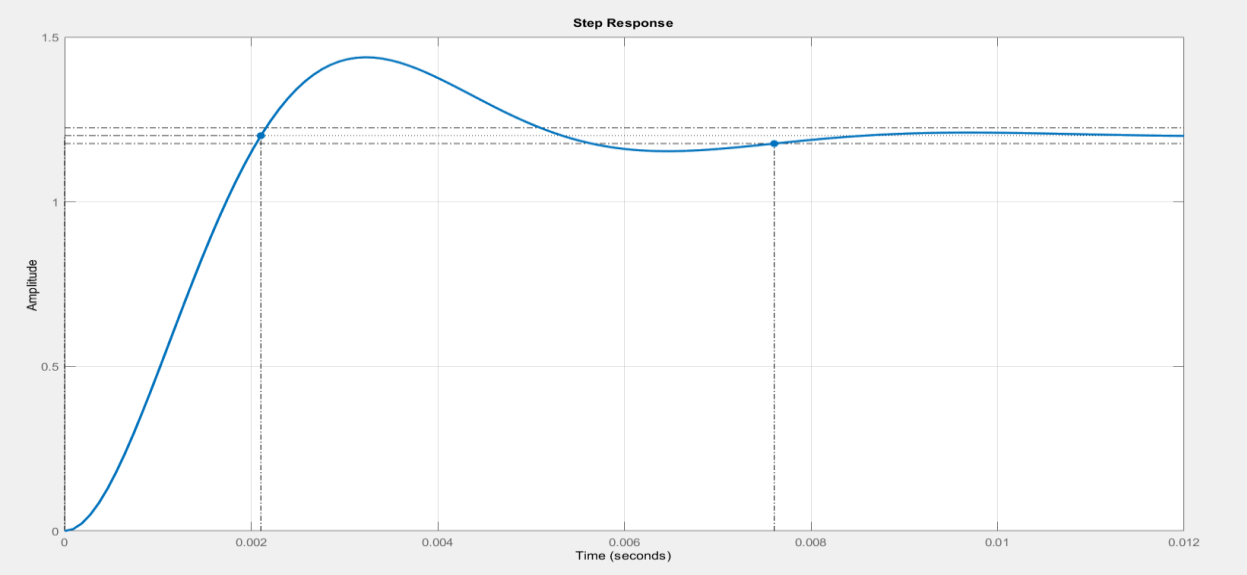


Figure 6: Continuous domain step response of the system in MATLAB

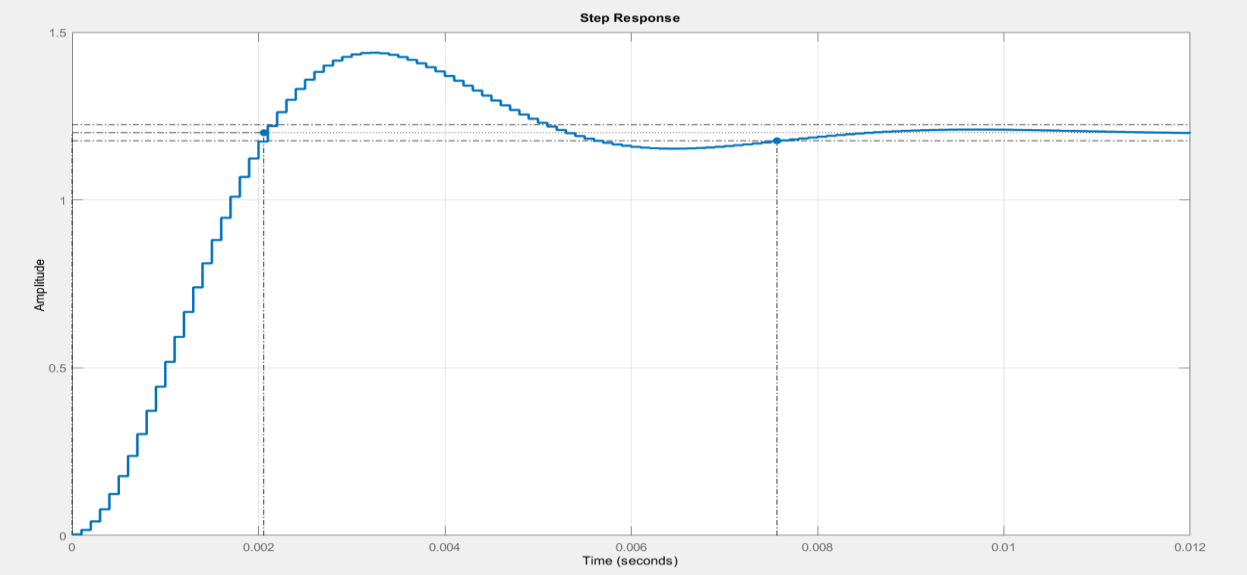


Figure 7: Discrete domain step response of the system in MATLAB

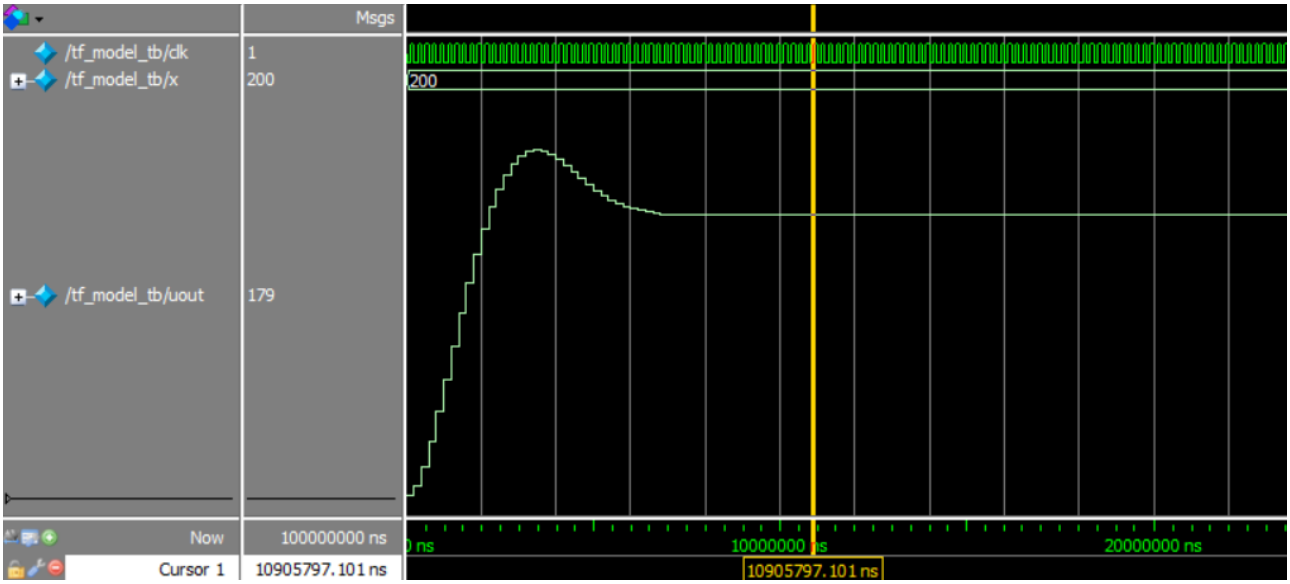


Figure 7: Step Response of TF in Verilog

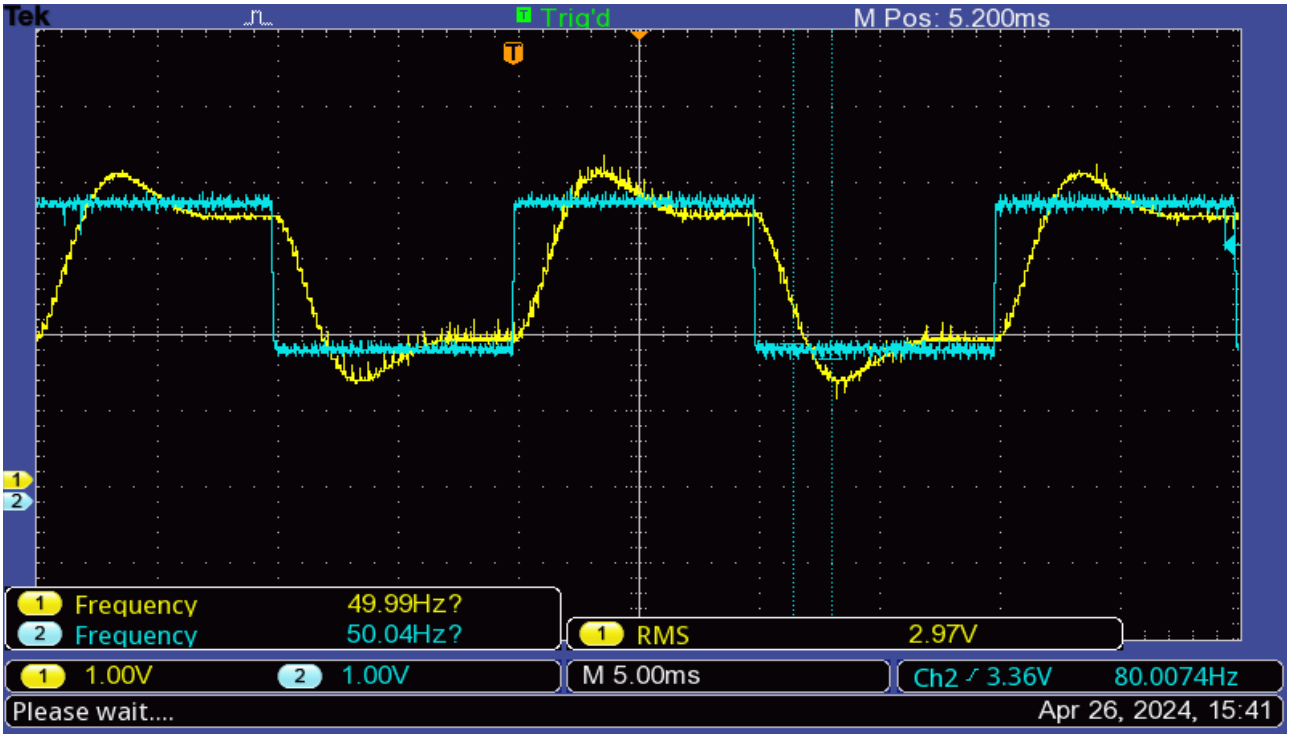


Figure 8: DAC output of the TF with a Square wave pulse

Table 1: Comparison of time domain specifications for different waveforms

Transfer Function	Continuous (Matlab)			Discrete (Matlab)			Discrete (ModelSim)		
	Rise Time (us)	Settling Time (us)	Steady State error (%)	Rise Time (us)	Settling Time (us)	Steady State error (%)	Rise Time (us)	Settling Time (us)	Steady State error (%)
G_1	107.34	213.21	2	100	210	2	100	205	3.17
G_2	2100	7610	0.042	2060	7560	0.0416	2000	7200	10.5

Error in Rise time for waveform simulated in Matlab and ModelSim:

$$e_1 = \frac{|107.34-100|}{107.34} * 100 = 6.83\%$$

Error in Settling time for waveform simulated in Matlab and ModelSim:

$$e_2 = \frac{|213.21-205|}{213.21} * 100 = 3.85\%$$

The deviation from the true value could be due to data quantization error which is the difference in the analog value and the closest available digital value.

However, the steady state error for the simulation of the transfer function $G_2(s)$ is very high as given in Table 1. So, we tested the transfer function $G_2(s)$, with more numbers of register bits (32 bits) and tried to measure the steady state error which then 0.001843 %. This is very less as compared to error in 8 bit register size.

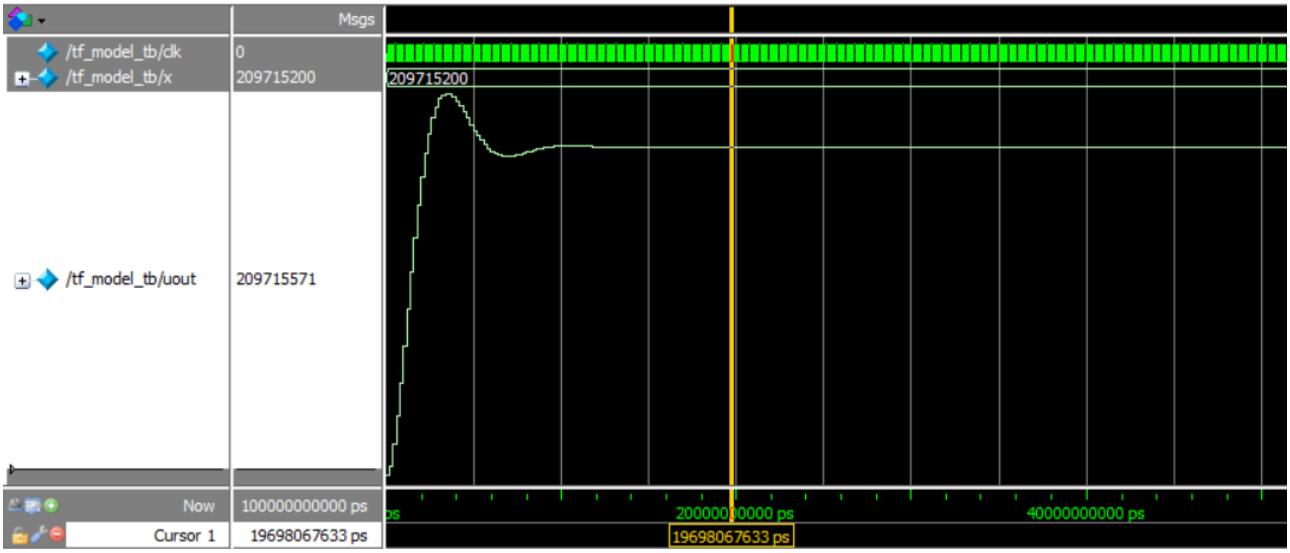


Figure 9: Step Response of TF with 32 bit register size

4.2 Frequency Response

Frequency response analysis of the transfer function involves studying how the system responds to different frequencies of input signals. This is crucial for understanding the behaviour of digital controllers implemented on FPGA (Field-Programmable Gate Array).

Second transfer function that is output voltage versus input voltage in Boost converter is taken for frequency response analysis.

$$G_2(s) = \frac{0.833}{5.798\text{x}10^{-7}s^2 + 5.807\text{x}10^{-4}s + 0.6939} \quad (4.7)$$

Experimental Setup

- **Selection of Input Signal Frequencies :** A range of input frequencies has been chosen covering both low and high frequencies to analyse the behaviour across the spectrum keeping in mind the frequency range up to the 3dB drop.

- **Input Signal Generation :** Sine wave of different frequencies has been generated digitally via Verilog code, where 360 samples of sine wave is taken for a period and data is sent at a particular sample rate to obtain the desired frequency.

- **Measurement of the Responses :** The implemented transfer function gives the output corresponding to a particular frequency sine wave input, which is taken out as an analog signal using a DAC card. The output is seen through the oscilloscope and corresponding amplitude and phase data is taken.

- **Frequency Response Data:** The experimental data has been listed in the table for different frequencies.

Table 2: Frequency Response Data

Frequency (Hz)	**V_{in}(V) (peak-peak)**	**V_o (V)(peak-peak)**	**Amplitude Ratio (dB)**	**dT (time lag b/w peaks)**	**Phase (degrees)**
30	2.92	3.44	1.4235	1	-10.8
40	2.92	3.24	0.9032	1.2	-17.28
50	2.92	3.2	0.7953	1.2	-21.6
60	2.92	3.16	0.6861	1.2	-25.92
80	2.92	3.4	1.3219	1.2	-34.56
100	2.92	3.4	1.3219	1.4	-50.4
120	2.92	3.32	1.1151	1.4	-60.48

150	2.92	3.36	1.2191	1.2	-64,8
160	2.92	3.36	1.2191	1.2	-69.12
180	2.92	2.6	--1.0082	1.5	-97.2
200	2.92	2.48	-1.4186	1.5	-108.0
210	2.92	2.24	-2.3027	1.6	-120.96
250	2.92	1.44	-6.1404	1.4	-126.0
300	2.92	1.04	-8.9670	1.3	-140.4
350	2.92	1.00	-9.3077	1.2	-151.2
400	2.92	1.00	-9.3077	1.1	-158.4
500	2.92	0.8	-11.2459	0.9	-162.0

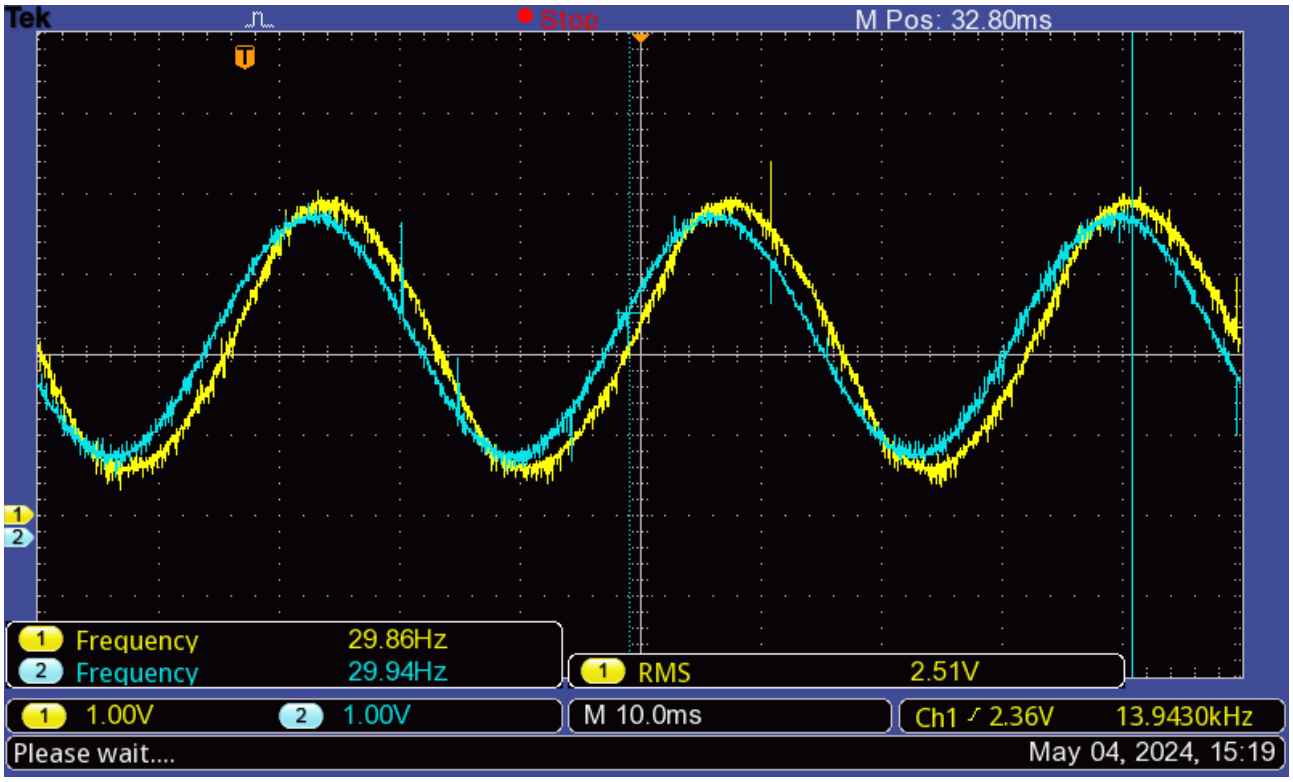


Figure 10: DAC Output with 30Hz sine wave

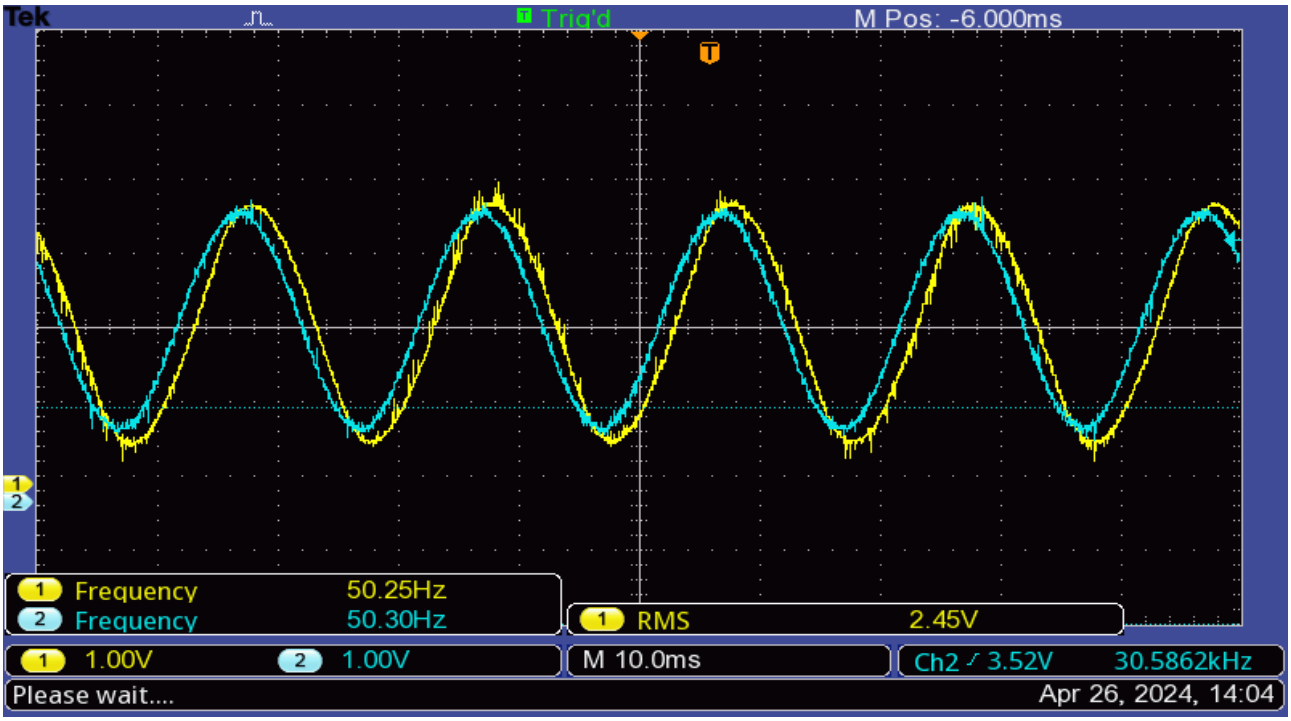


Figure 11: DAC Output with 50Hz sine wave

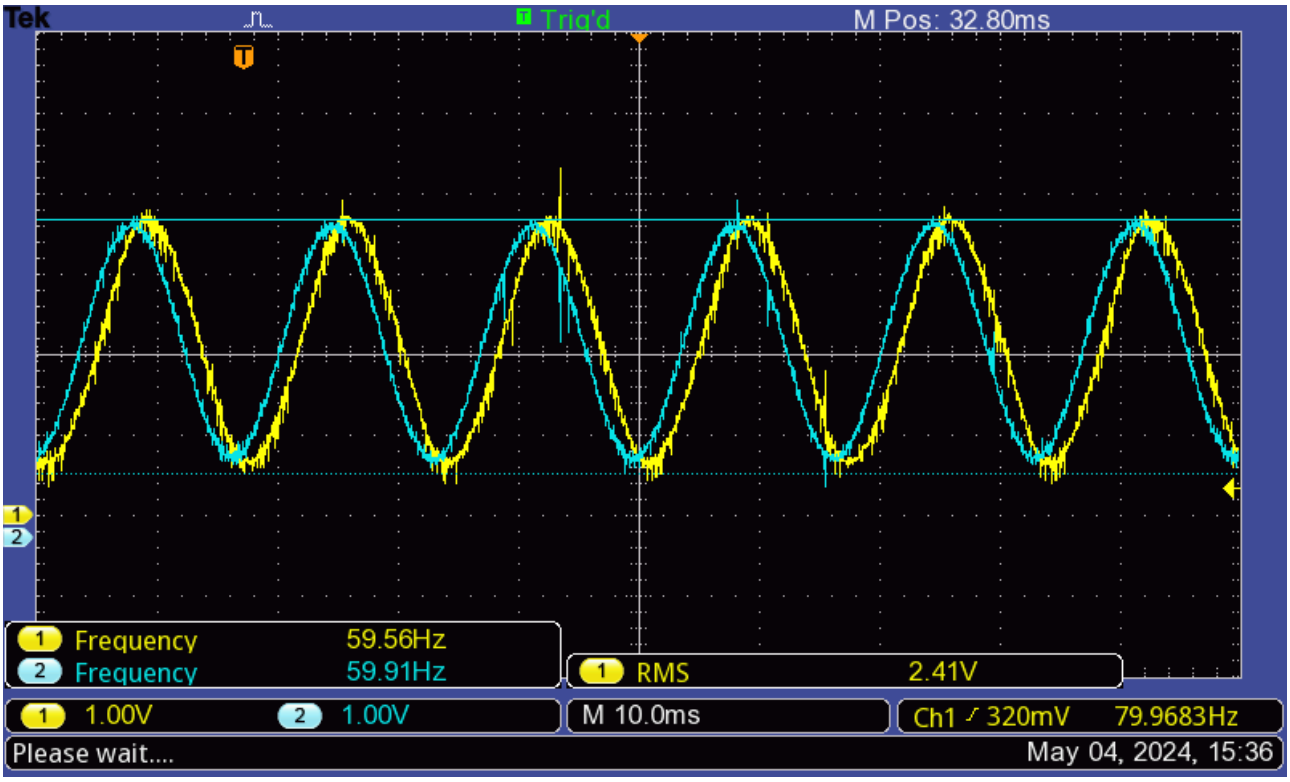


Figure 12: DAC Output with 60Hz sine wave

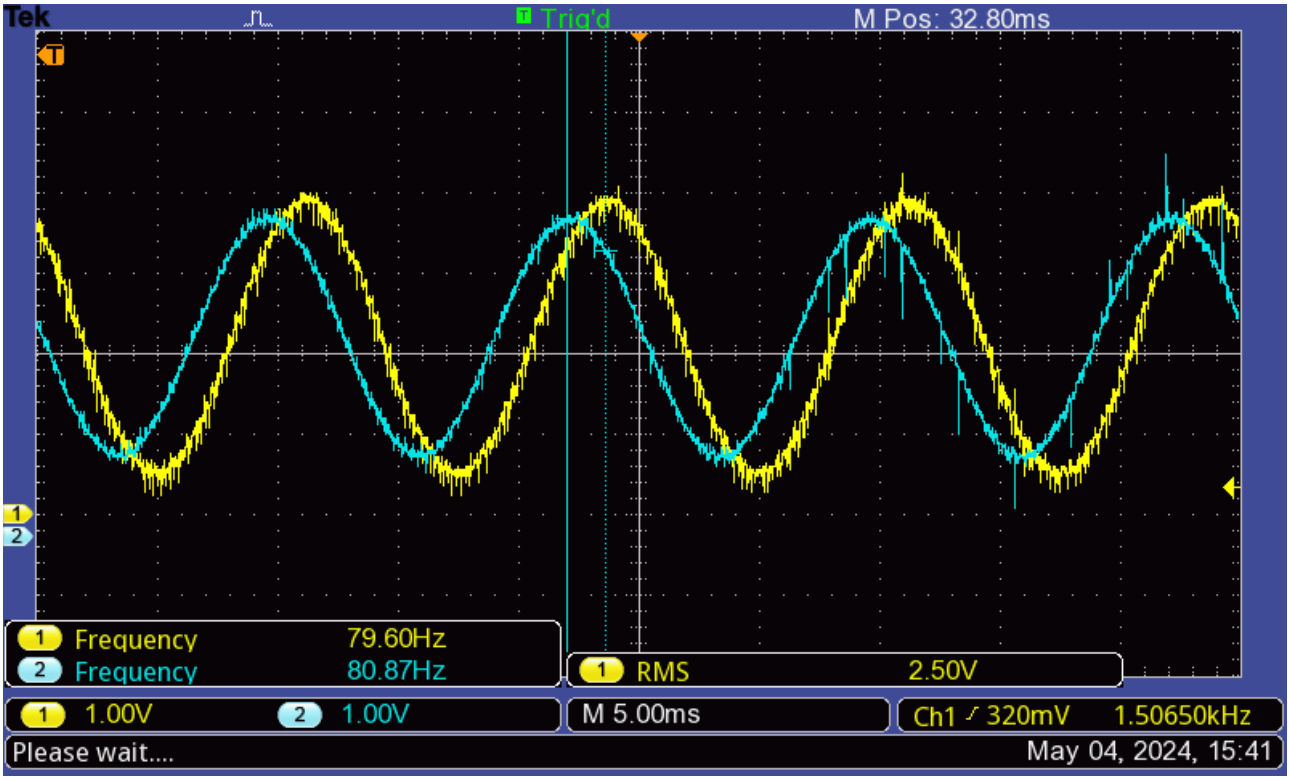


Figure 13: DAC Output with 80Hz sine wave

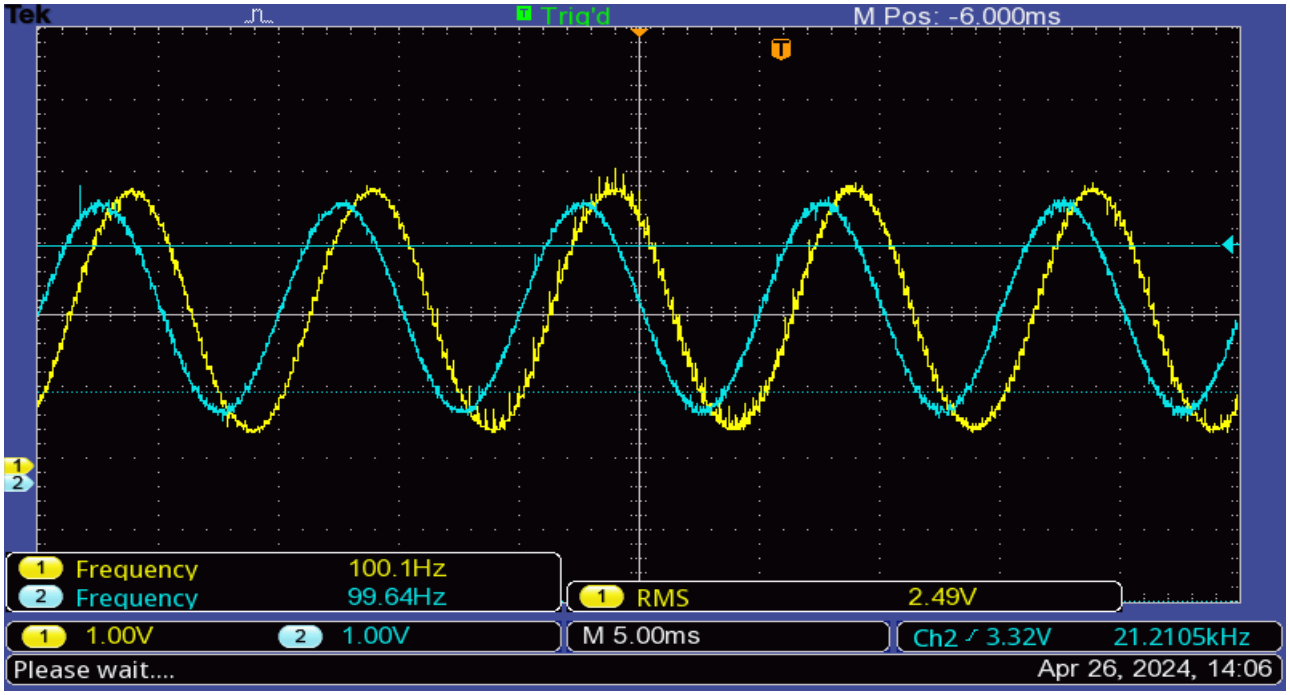


Figure 14: DAC Output with 100Hz sine wave

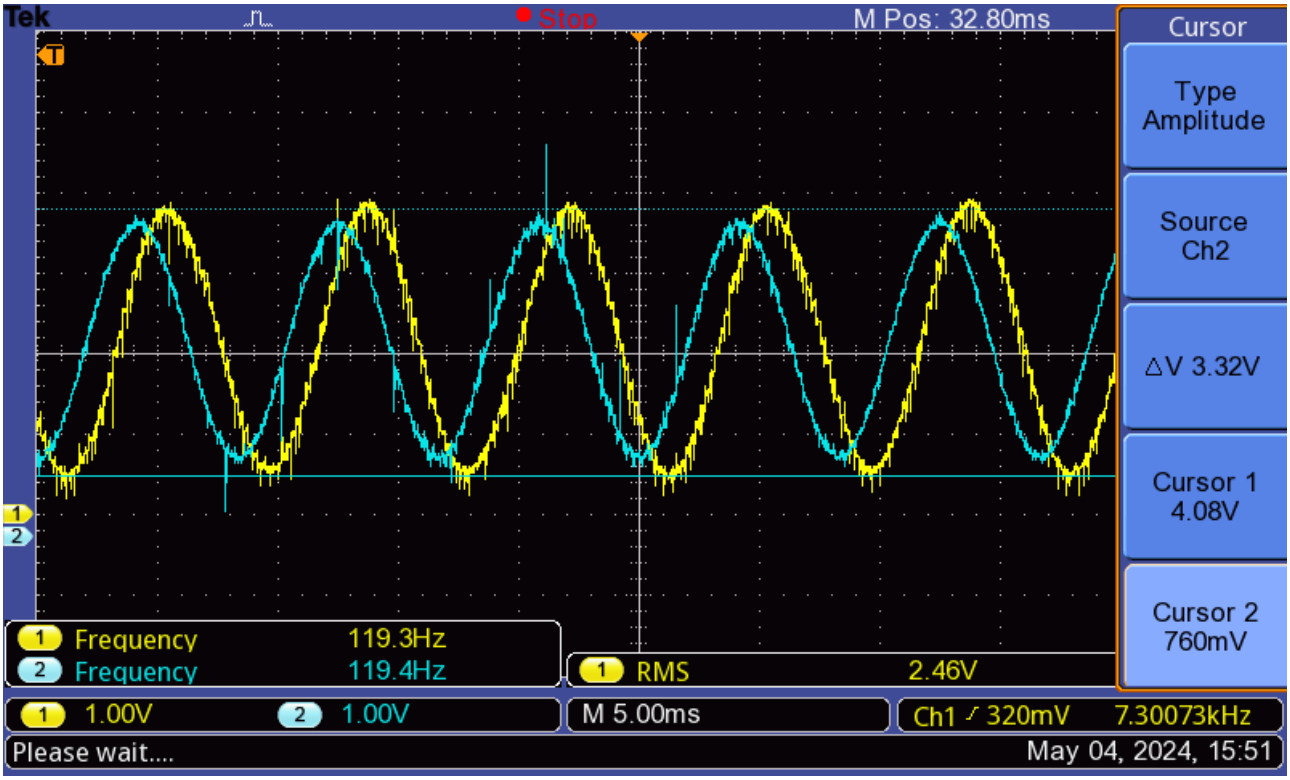


Figure 15: DAC Output with 120Hz sine wave

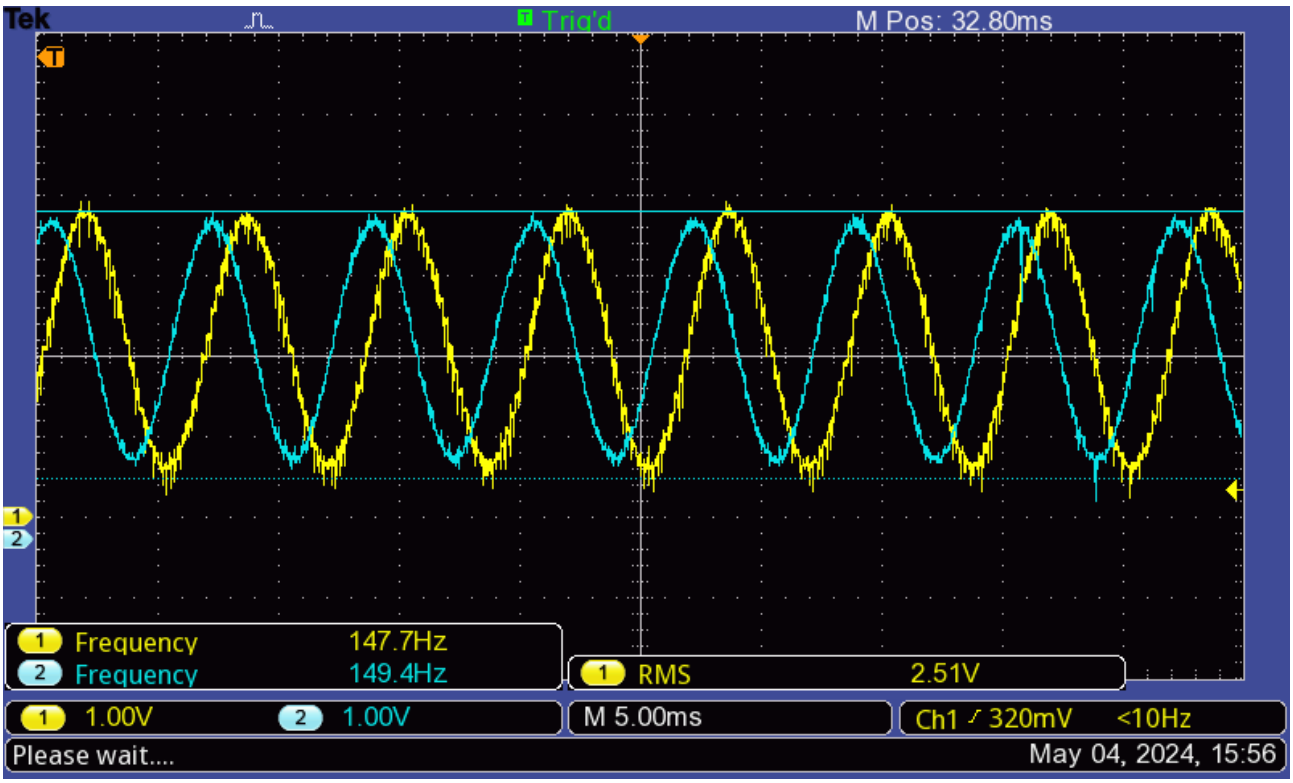


Figure 16: DAC Output with 150Hz sine wave

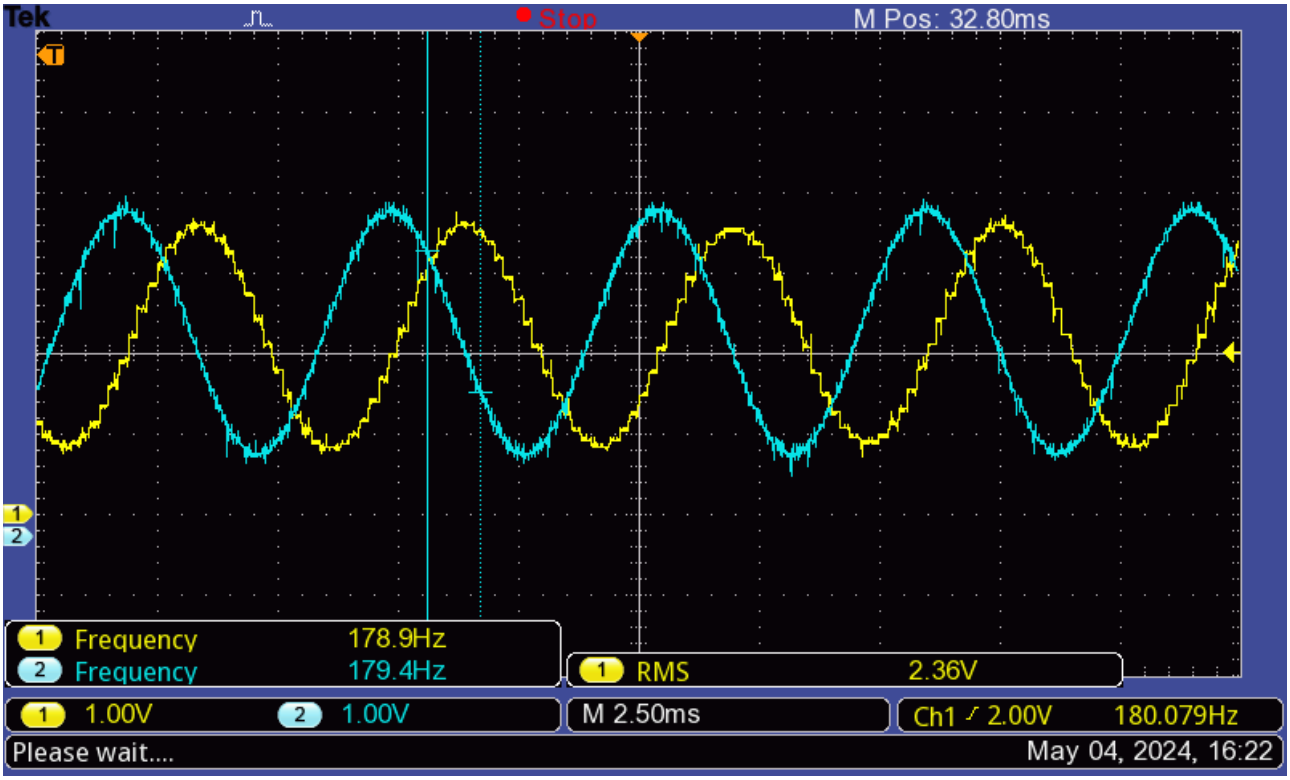


Figure 17: DAC Output with 180Hz sine wave

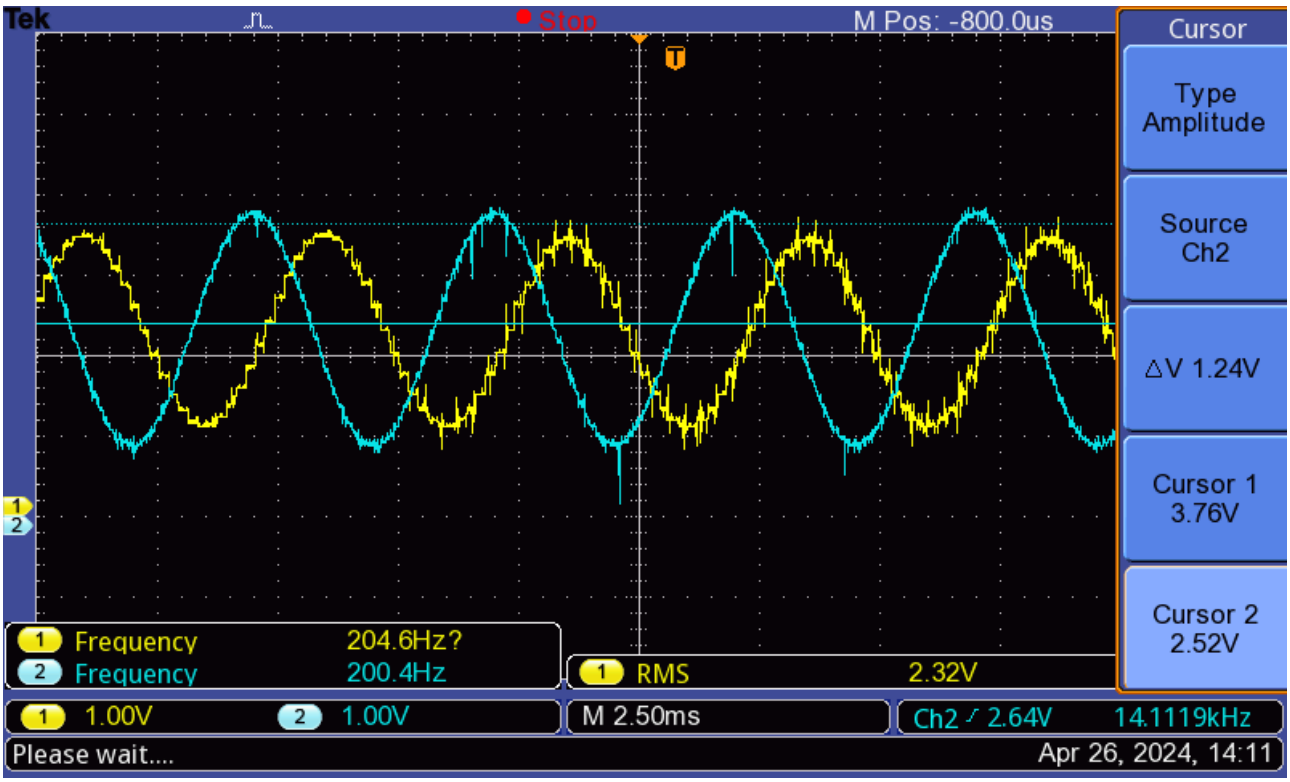


Figure 18: DAC Output with 200Hz sine wave

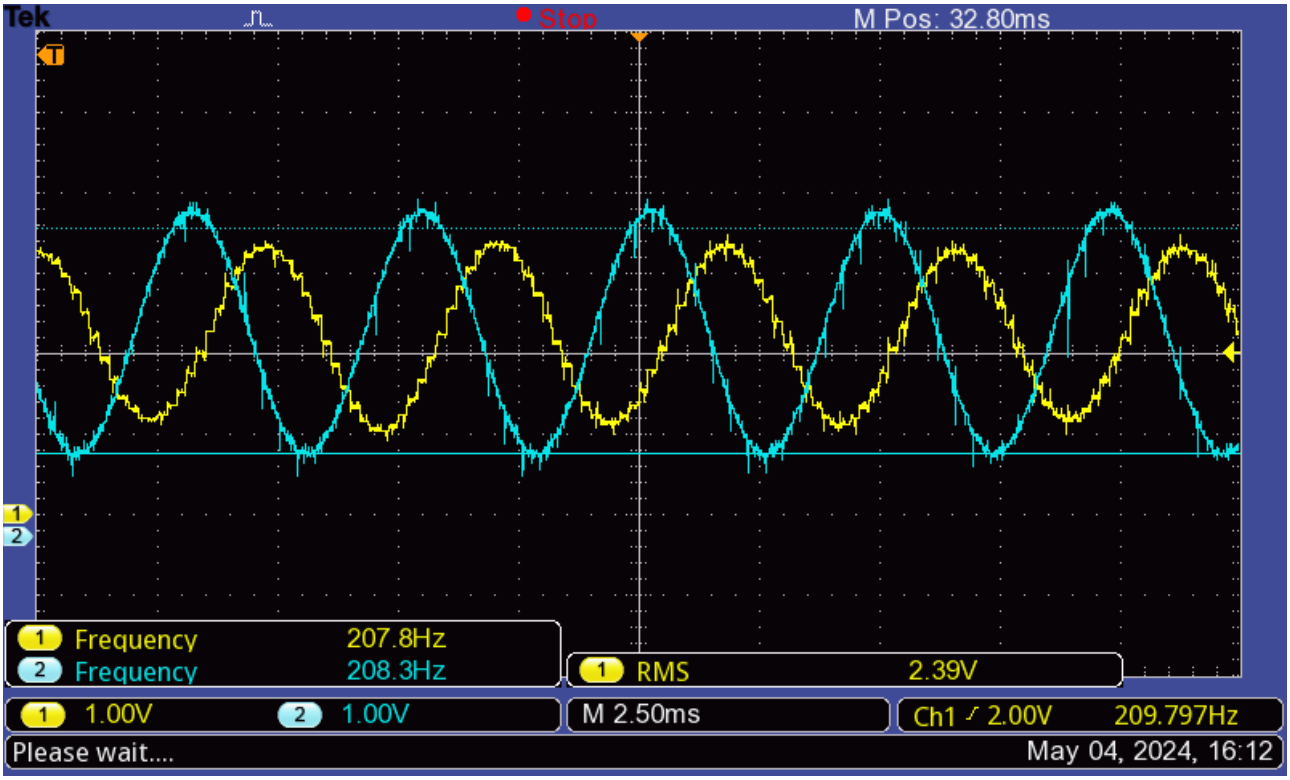


Figure 19: DAC Output with 210Hz sine wave

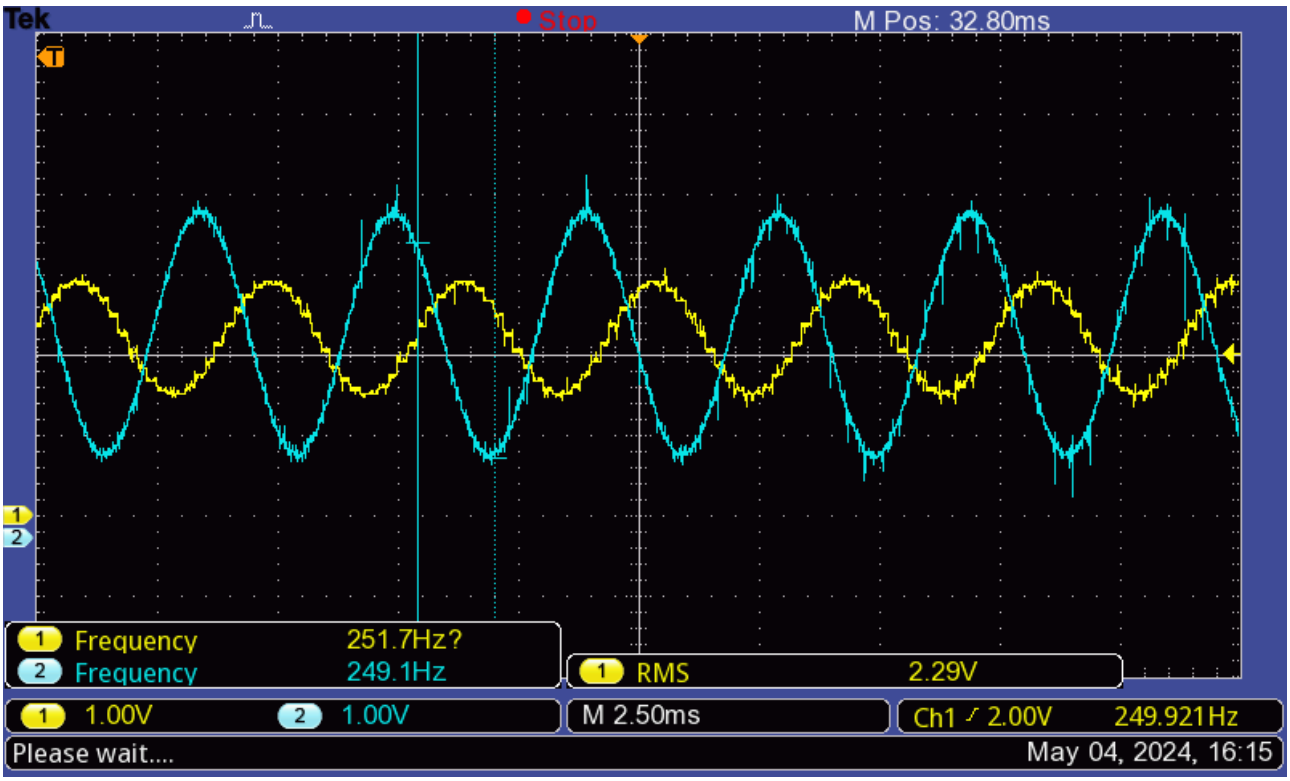


Figure 20: DAC Output with 250Hz sine wave

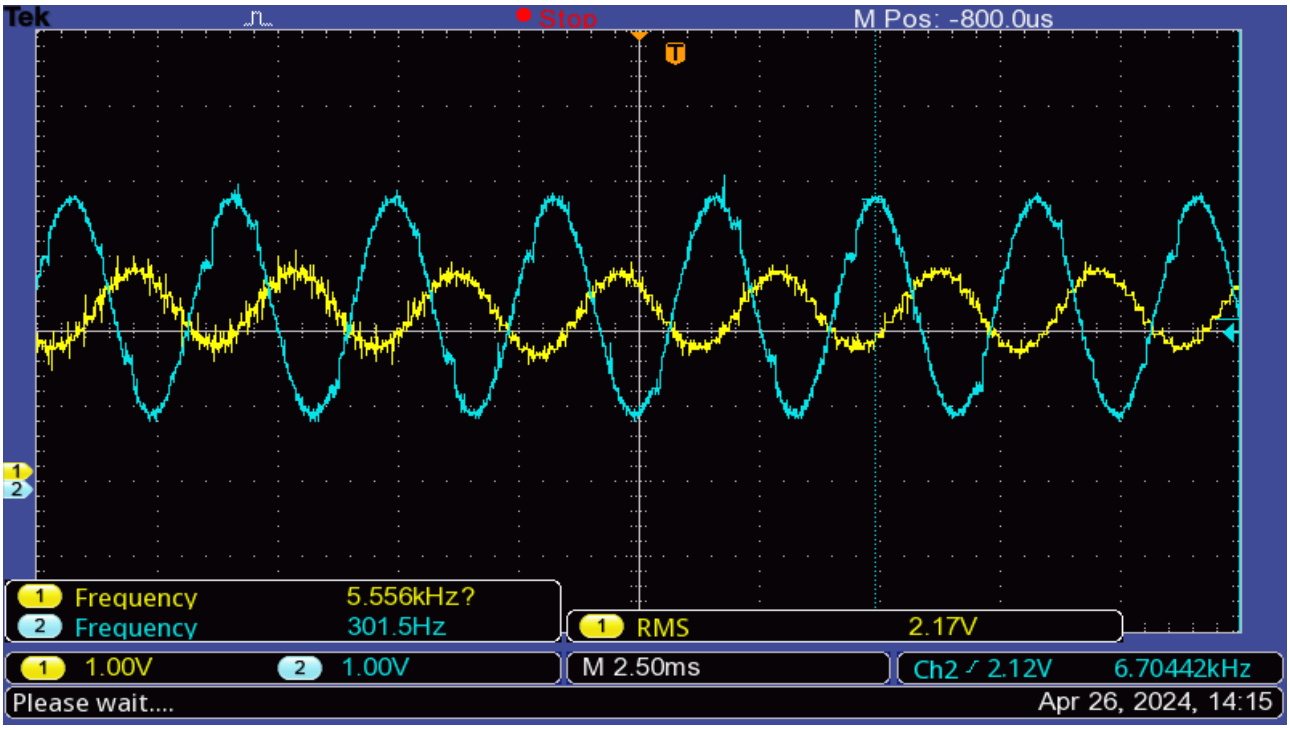


Figure 21: DAC Output with 300Hz sine wave

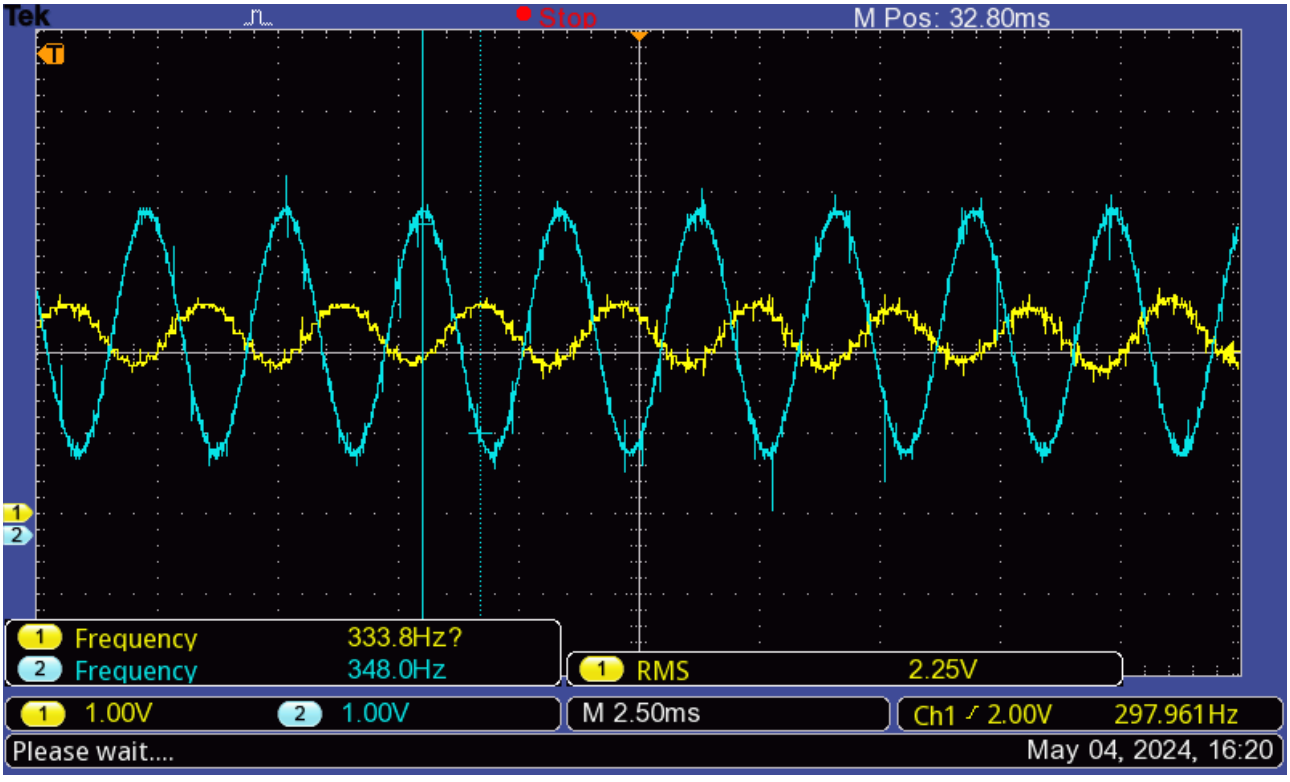


Figure 22: DAC Output with 350Hz sine wave

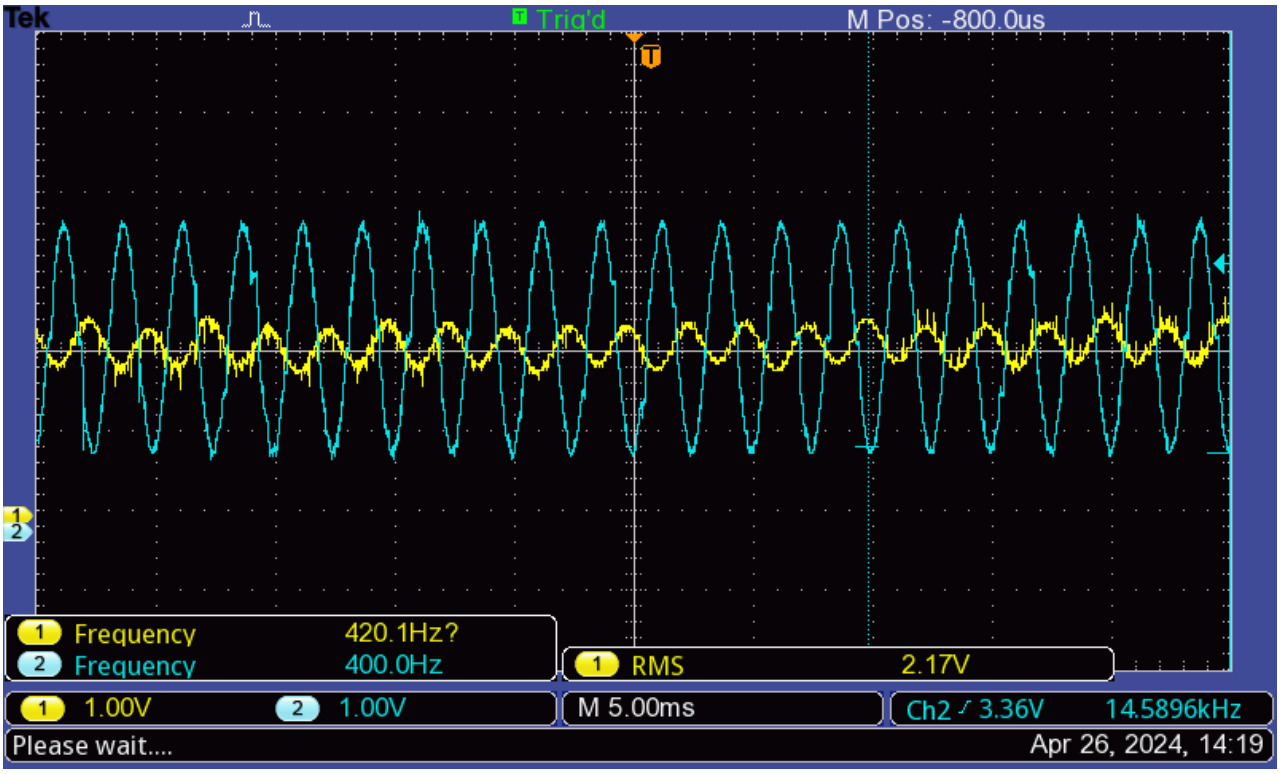


Figure 23: DAC Output with 400Hz sine wave

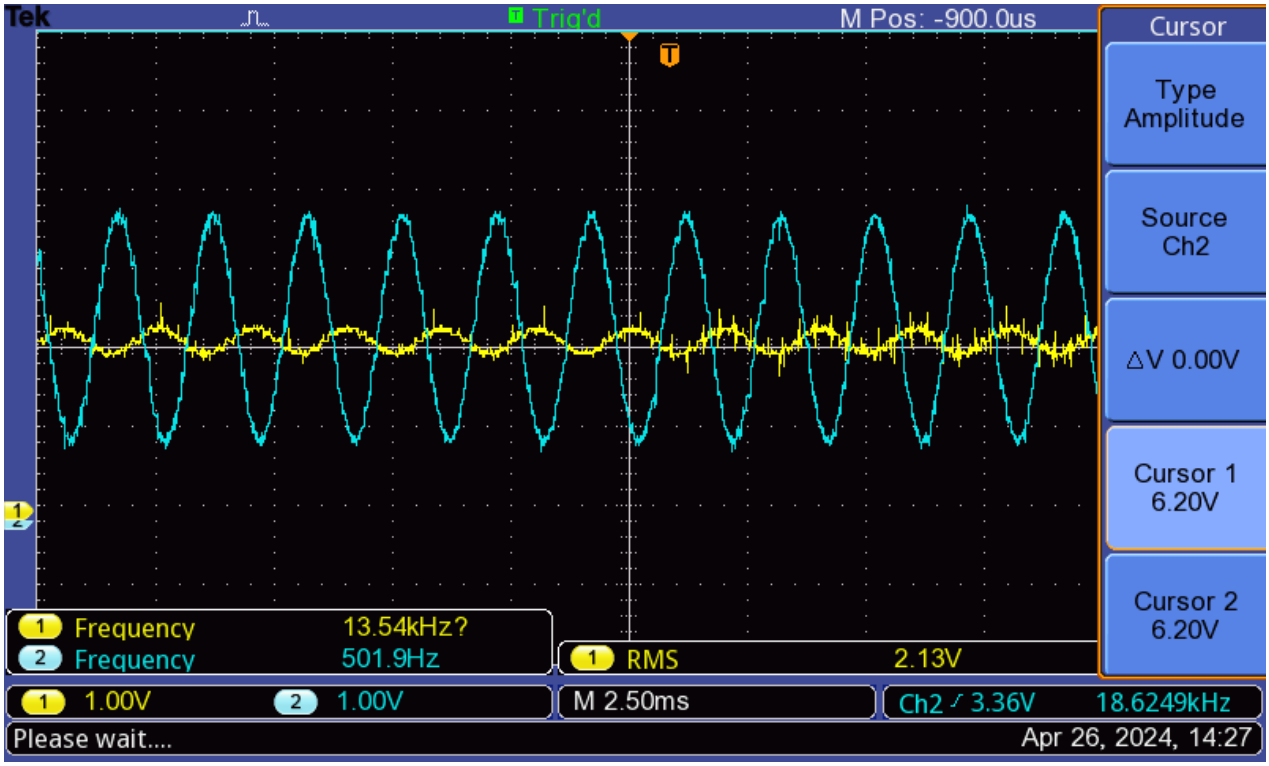


Figure 24: DAC Output with 500Hz sine wave

- **Analysis and Interpretation :** The frequency response, represented by a Bode plot, is generated using MATLAB software, encompassing a comprehensive frequency spectrum. Subsequently, experimental data is superimposed onto the same semi-log graph to facilitate direct comparison with the Bode plot.

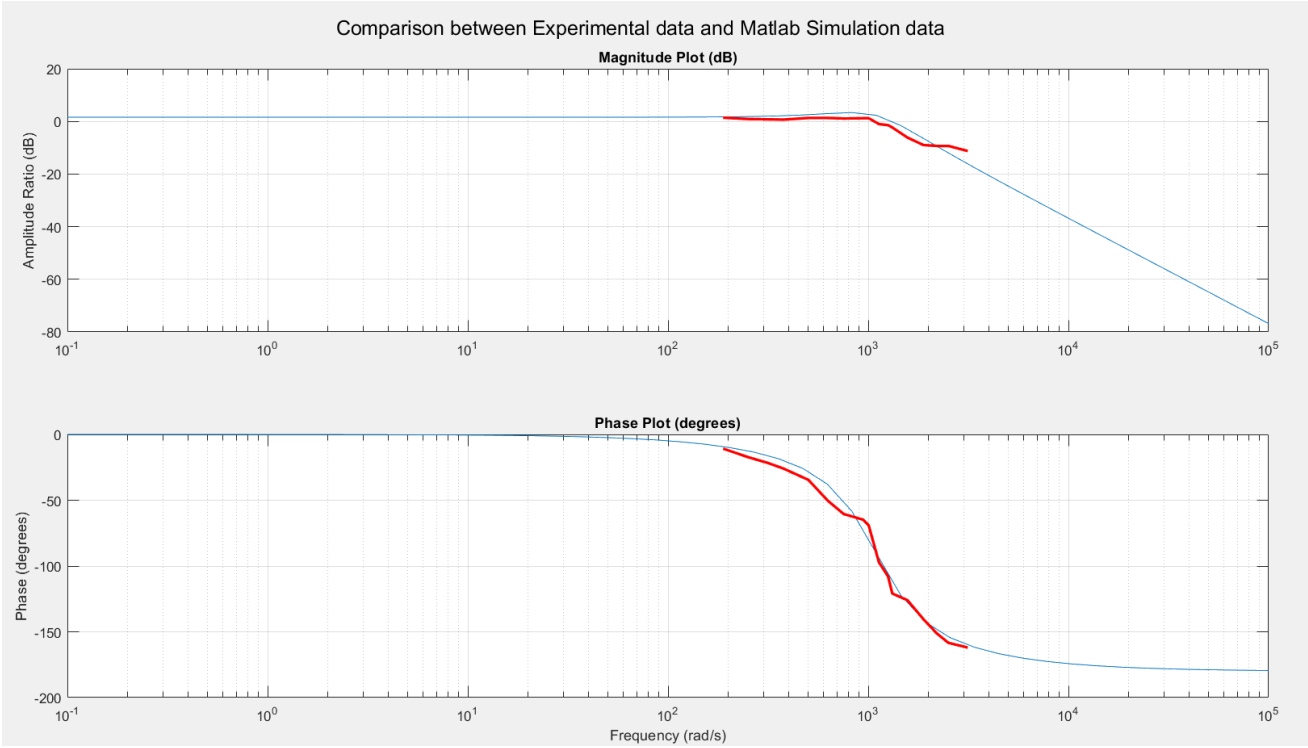


Figure 25: Bode plot for Experimental data (in Red) and Matlab Simulation Data (in Blue)

- **Conclusion :** The frequency response of the transfer function measured in simulation shows almost same behaviour as the expected bode plot. The deviation from the expected value could be attributed to the less samples (in this case 360 samples per period) of the sine wave taken as the input to the transfer function.

5. I2C controller code in Verilog

To see the output generated by the FPGA on an oscilloscope, it needs to be converted into analog signal using a DAC. For this purpose, MCP4725, an I2C based DAC is used. To interface it with the FPGA, an I2C controller core needs to be developed in Verilog as per the standards of the protocol.

5.1 I2C protocol:

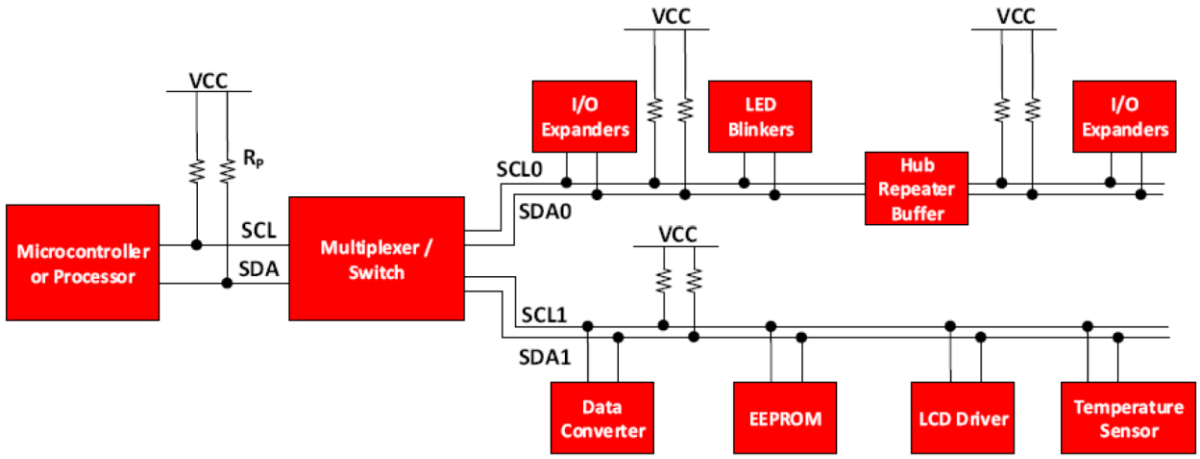


Figure 26: I2C Interface. Source: Adapted from [5]

I2C stands for Inter Integrated Circuit and is a serial communication protocol. It is a 2-wire communication protocol, using just two signals, namely SDA and SCL to transfer digital data. Following are the key features of I2C protocol:

i. All the devices on the bus are connecting to only two communicating lines (SDA and SCL).
ii. It is a half-duplex, bi-directional communication protocol.
iii. The communication is always initiated by the master.
iv. It can support multiple masters and multiple slaves on the bus.
v. It is very easy to implement compared to other communication protocols because of less complexity.
vi. It incorporates an acknowledge bit which increases its robustness and help in error free communication.
vii. The slave devices connected to the bus are identified by a 7-bit address.

The I2C communication uses a ‘START’ sequence and a ‘STOP’ sequence to signal the beginning and the end of the communication.

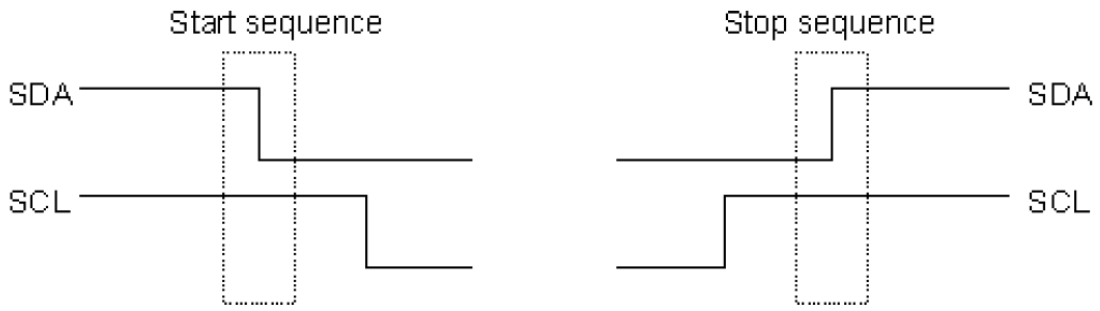


Figure 27: 'START' and 'STOP' sequence for I2C Protocol. Source: Adapted from [5]

The I2C bus provides the option of writing as well as reading from the slave. Since, for the purpose of this project, the DAC is acting as the slave device, the data only needs to be written to the DAC register by the master. So, to reduce the complexity of the code for the controller core, only the write functionality to the slave has been implemented. The speed of communication can be selected by modifying the input parameters to the code.

Writing to a Slave on I2C bus:

i. First, the start sequence is generated when the master pulls the SDA low while SCL is high.
ii. The 7-bit slave address and the R/W bit is sent serially. The R/W bit is kept '0' indicating write operation.
iii. Then the master waits for the acknowledge bit by the slave. If the bit is '0', it denotes that the slave address has been identified and it is ready for write operation. If the bit is '1', then it indicates the communication has failed.
iv. After successful acknowledge by the slave, the internal register number is sent to which the data needs to be written.
v. The master again waits for the acknowledge bit to ensure that the sent register location has been received by the slave.
vi. After this, the master sends 8-bit data to the slave.
vii. If more data needs to be sent, then the master continues sending the data in serial 8-bit chunks.
viii. If there is no data to be sent, then the stop sequence is generated by the master by pulling the SDA high while SCL is high.

Testing Procedure:

The testing of I2C core in Verilog was done using a testbench in Verilog. The testbench initializes the I2C core module and feeds it the necessary signals such as the clock, the address, the data to be written, etc. The simulation was done in ModelSim which generates a waveform for the output signals so that the results could be verified.

MCP4725 DAC Module:

The I2C DAC module MCP4725 is used to see the output of the transfer function on the oscilloscope. The datasheet of the device describes the address as well as its theory of operation.

"The MCP4725 device address contains four fixed bits (1100 = device code) and three address bits (A2, A1, A0). The A2 and A1 bits are hard-wired during manufacturing, and A0 bit is determined by the logic state of A0 pin. The A0 pin can be connected to VDD or VSS, or actively driven by digital logic levels." [6]

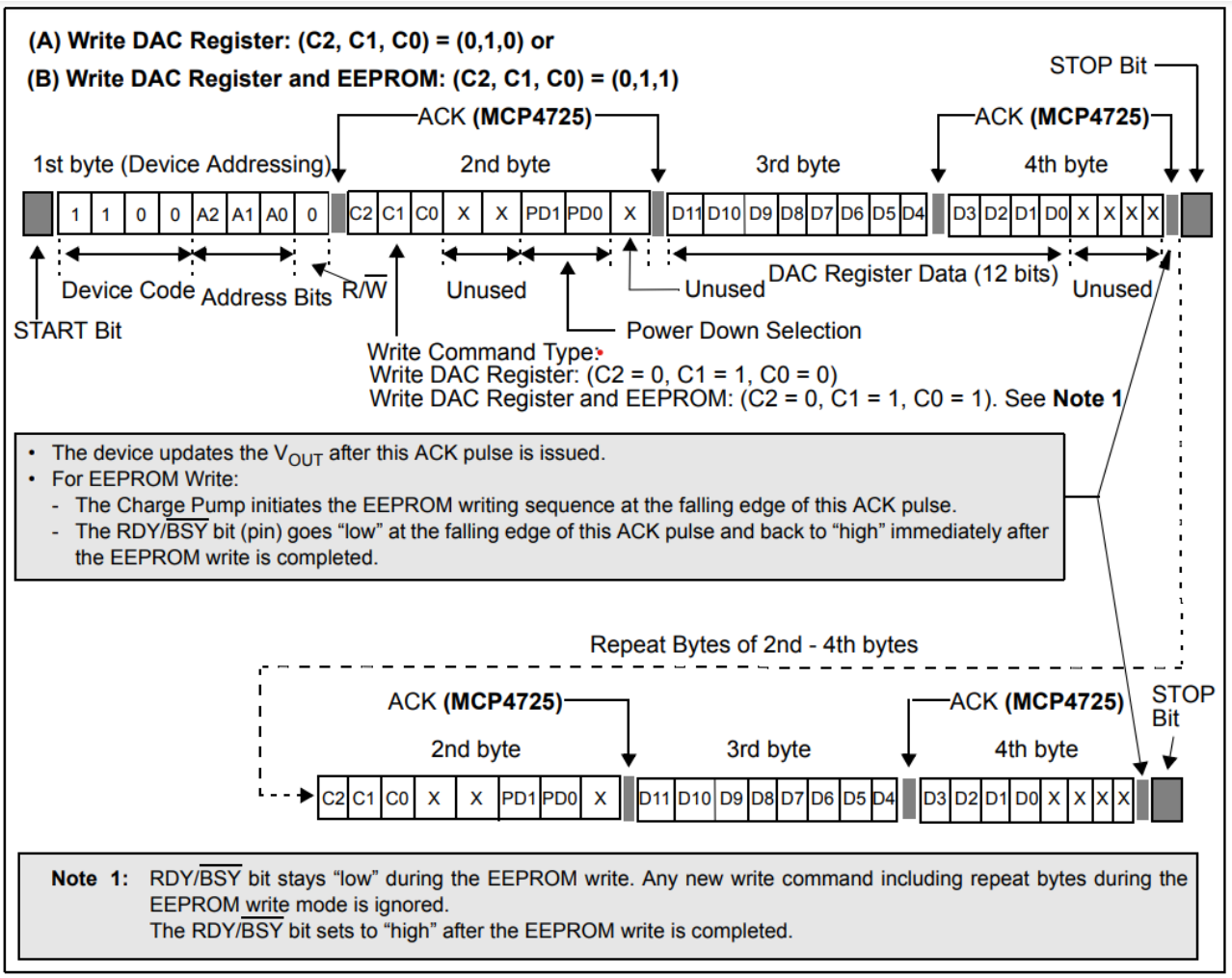


Figure 28: Write Commands for DAC Input Register and EEPROM. Image Source[6]

The data packets from the FPGA to the DAC module using the SDA and SCL lines is shown in the figure below.

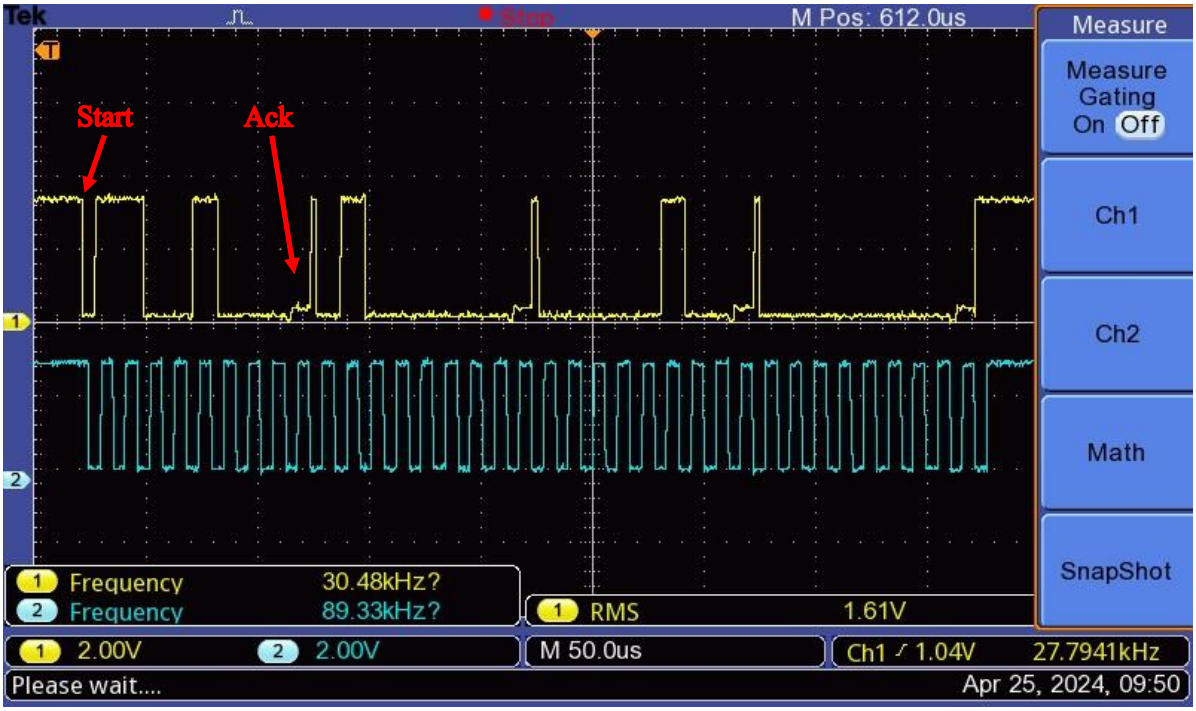


Figure 29: SDA (in yellow) and SCL (in blue) lines from the FPGA to the DAC

The DAC Code was tested using a 50Hz sinusoidal signal with 360 samples per period. The speed of the I2C clock was set at 100kHz. The figure below shows the output after the digital to analog conversion.

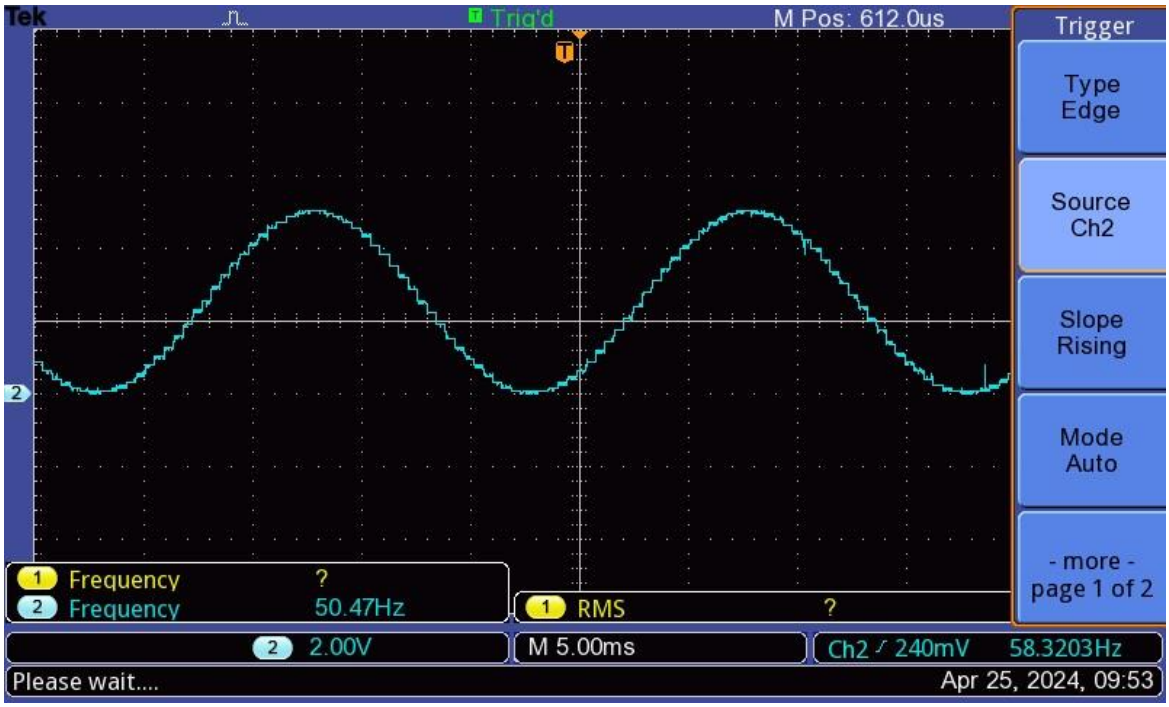


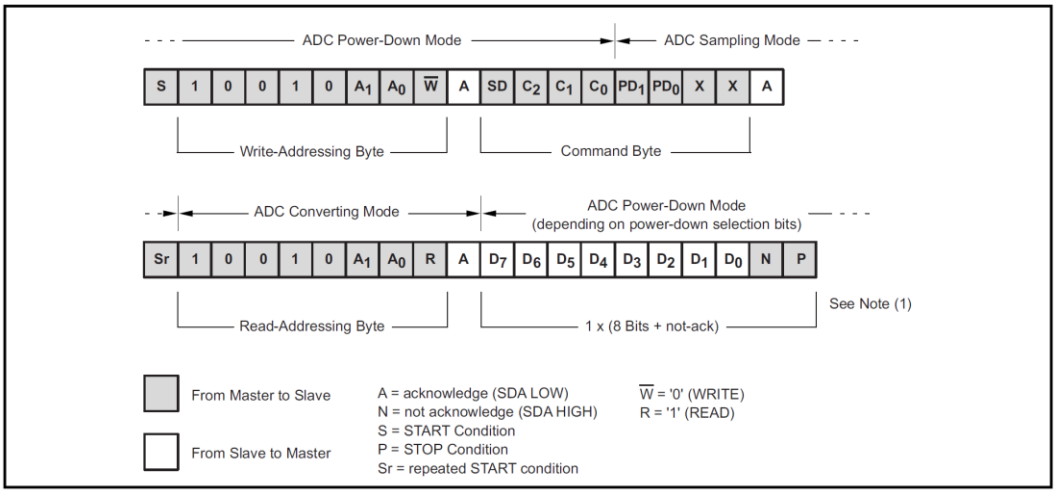
Figure 30: DAC output at 100kHz I2C bus speed

Reading from a Slave on I2C bus:

i. First, the start sequence is generated when the master pulls the SDA low while SCL is high.
ii. The 7-bit slave address and the R/W bit is sent serially. The R/W bit is kept '0' indicating write operation.
iii. Then the master waits for the acknowledge bit by the slave. If the bit is '0', it denotes that the slave address has been identified. If the bit is '1', then it indicates the handshake has failed.
iv. After successful acknowledge by the slave, the internal register number is sent from which the data needs to be read or any command word which is required to initialise the device.
v. The master again waits for the acknowledge bit to ensure that the read register location has been received by the slave or the command word has been identified.
vi. A repeated start bit is generated on the next pulse.
vii. After this, the master hands over the SDA line to the slave which then begins transmitting a 8-bit data packet.
viii. The master needs to acknowledge whether the data has been read or not. If more data needs to be read then the master pulls the SDA line low on the acknowledge pulse otherwise it leaves the SDA line in high impedance state indicating a NOT ACK. This means that no further data needs to be read.
ix. The stop sequence is generated by the master by pulling the SDA high while SCL is high and the communication is terminated.

ADS7830 ADC Module:

The ADS7830 is an 8-bit, 8-channel, Successive Approximation Register (SAR) I2C A/D converter. The relevant data regarding the interfacing and operation of this device can be found in its datasheet. The device communicates with the FPGA using the two I2C lines, namely the SDA and SCL lines. *"The first five bits (MSBs) of the slave address are factory pre-set to 10010. The next two bits of the address byte are the device select bits, A1 and A0. Input pins (A1-A0) on the ADS7830 determine these two bits of the device address for a particular ADS7830."* [7]



(1) To secure bus operation and loop back to the stage of write-addressing for next conversion, use repeated START.

Figure 31: Read Operation for the ADC. Image Source: [7]

The data packets on the SDA and SCL lines as observed on the oscilloscope for the ADC communicating with FPGA is given below.

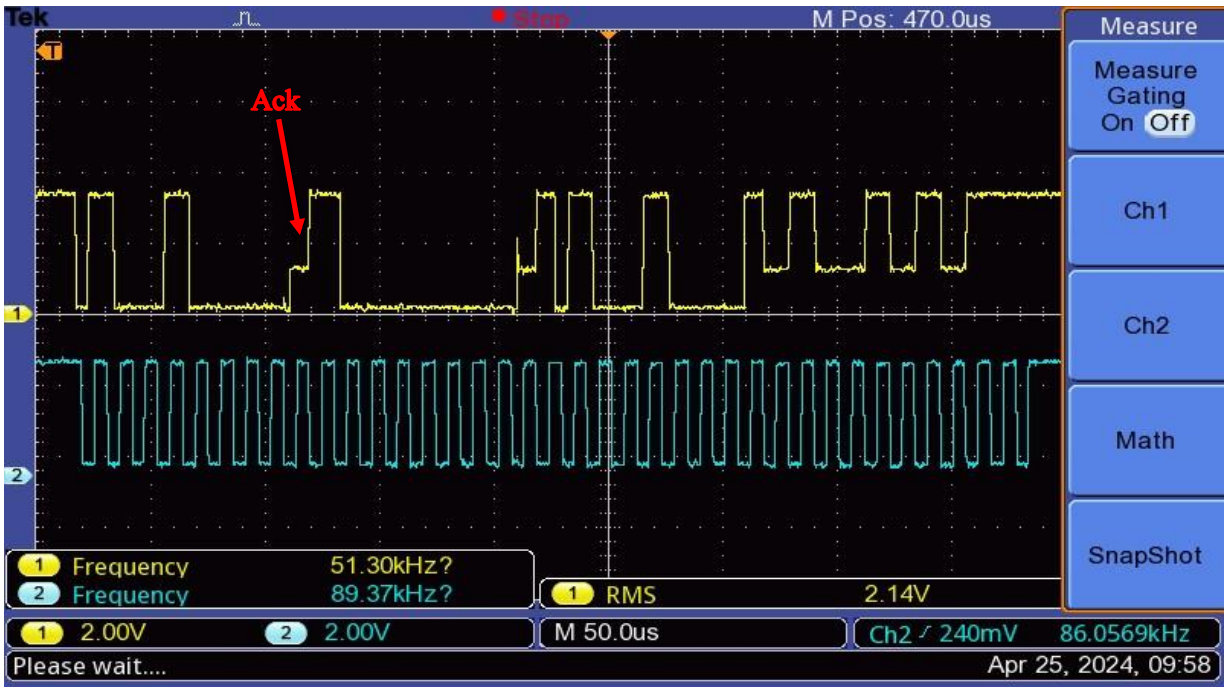


Figure 32: SDA (in yellow) and SCL (in blue) lines from the FPGA to the ADC

In Fig. 22 above, it can be observed that the ADC module is recognizing its address (which is 0x90, including write bit at the end) and trying to acknowledge, but the pull down level of the SDA line remains at 1.8V whereas it should have been at around 0.4V or less. As a result, when it is trying to send the digital data after conversion, the bits which are supposed to indicate logic level '0' are pulled down to 1.8V, which is very high and will not get registered as low level. Hence, in effect the data sent by ADC module is a sequence all 1s.

When the input channel data is varied, a change in the output of the ADC can be noticed which means that the conversion is correct but during transmission of data due to insufficient pull down of the SDA line, it appears to be wrong. Despite continued efforts to troubleshoot the problem, no solution to this problem has been achieved as of the time of drafting of this report.

6. Conclusion

In conclusion, the report documents the implementation of a digital controller on the FPGA Cyclone II/Cyclone IV module, incorporating I2C DAC and ADC functionalities. The transfer function/controller implemented has undergone testing, encompassing both time and frequency domain analyses. Through evaluation of time response characteristics and verification of frequency response using BODE plots, the controller design has been validated. These testing procedures provide valuable insights into the dynamic behavior and stability of the system, further enhancing the credibility and applicability of the implemented digital control solution.

Conversely, challenges encountered during the integration and operation of the ADC, particularly related to unresolved pull-down issues, have been documented. Despite concerted efforts, the ADC has not yet achieved functionality as intended. The inclusion of detailed timing diagrams in the report serves to provide a comprehensive overview of observed phenomena and aids in identifying potential avenues for resolution.

The findings and insights presented in this report contribute significantly to the body of knowledge surrounding digital control implementations on FPGA platforms. Moving forward, addressing the identified issues with the ADC will require further analysis and iterative refinement of design and implementation methodologies.

In summary, while the project has yielded notable successes in the operation of the DAC and the testing of the transfer function/controller, it has also presented valuable opportunities for learning and improvement, particularly in addressing the complexities of ADC integration.

7. References

[1] S. Dalapati, "A Control Systems Laboratory Experiment on Transfer Function Emulation", Proceedings of 2020 Applied Signal Processing Conference (ASPCON), 2020.

[2] P. Trujillo. "Implementation of an IIR filter on FPGA from scratch" hackster.io. https://www.hackster.io/pablotrujillojuan/implementation-of-an-iir-filter-on-fpga-from-scratch-4b8539

[3] C. E. Cummings, "Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill!" *SNUG-2000*, San Jose, CA, USA, 2000.

[4] K. Ogata. "Mathematical Modelling of Control Systems" in *Modern Control Engineering*, 5th ed., Pearson, 2015

[5] N. Nise. "Digital Control Systems" in *Control System Engineering*, 7th ed., Wiley, 2015

[6] Prof. D Ganguly. "Introducing the I^2C Bus" Lecture Notes

[7] Microchip Technologies Inc. *MCP4725 12-Bit Digital-to-Analog Converter with EEPROM Memory in SOT-23-6 Datasheet* (2009). [Online] Available: https://ww1.microchip.com/downloads/en/devicedoc/22039d.pdf

Appendix : A

Controller Code in Verilog:

```
module tf_model(
  input clk,
  input [7:0] x,
  output [7:0] uout
);

reg[7:0] pipe1;
reg[7:0] pipe2;
reg[7:0] pipe3;
reg[7:0] pipe4;

localparam b0 = 8'h01;
localparam b1 = 8'h03;
localparam b2 = 8'h01;
localparam a1 = 8'hE3;
localparam a2 = 8'h68;
reg [15:0] y;


assign uout = y>>1;

always @(posedge clk) begin
  if(pipe3)
    pipe4 <= pipe3;   //y[n-2]
  else
    pipe4 <= 0;
end

always @(posedge clk) begin
  if(uout)
    pipe3 <= uout;   //y[n-1]
  else
    pipe3 <= 0;
end

always @(posedge clk) begin
  if(pipe1)
```

```
    pipe2 <= pipe1;   //x[n-2]
  else
    pipe2 <= 0;
end

always @(posedge clk) begin
  pipe1 <= x;   //x[n-1]
end

always @(negedge clk) begin
   y =  (b1 * pipe1) + (b0 * x) + (b2 * pipe2) + (a1 * pipe3) - (a2 * pipe4);
    y = y >> 7;
   $display("%d",y);
  end

endmodule
```