A Hardware Accelerator for an Audio Auto-correlator

Nicholas Wolf
University of Auckland Computer Engineering
nwol626@aucklanduni.ac.nz

Abstract

Autocorrelation is a mathematical quantification of a time series' similarity to its lagged self [1]. Evaluating the autocorrelation of a time series allows analysts to determine the degree of relation between data points and the predecessors in their vicinity [2]. Autocorrelation is a powerful tool in signal processing applications and is predominantly used in the field of statistical analysis to analyze trends in specialized data. Additionally, autocorrelation may be employed in embedded audio processing applications to serve as an arbiter for the relativity of a continuous signal. However, in high bandwidth computing environments, an unadorned CPU may not provide the processing ability to effectively use time while inferring the autocorrelation of a continuous signal. For that reason it is of interest to apply hardware accelerators as a relief to the bottleneck caused by autocorrelation within embedded real-time systems.

Review of Literature

The concept of autocorrelation was proposed in the 1920′s by British statistician George Udny Yule as a means of evaluating the quantitative consequences of data and their relations [3]. Similarly to other signal processing algorithms, autocorrelation maintains the integrity of the analyzed signal while highlighting significant features. Autocorrelation specifically amplifies the energy of a time series. For its ability to provide insight regarding the statistical significance of trends within data, autocorrelation is commonly used to evaluate market trends. Analysts that use autocorrelation can relate previous trends to current data, discover significance, and make predictions regarding future expected market values [4]. Additionally, due to the capability for the autocorrelation function to highlight repeating patterns in data, it is employed in electrical signal processing applications which encounter significant degrees of noise obstruction such as audio processing [5]. Finally, the autocorrelation function is well received in digital design due to its compatibility with parallelism [6].

$$R[k] = \sum_{\{n=k\}}^{\{N-1\}} x[n]x[n-k]$$

Equation 1: Autocorrelation of time series x at lag k

Motivation

Autocorrelation is a powerful tool when integrated in an embedded signal processing environment as it provides a quantified relationship between a time series and itself. This information can be used to identify trends, anomalies, and patterns within data. Moreover, in the context of an audio processing application, an autocorrelation function can identify repetitive patterns in data. Detecting patterns within audio data is a valuable asset to audio applications such as speech recognition and music analysis, both common examples of embedded real-time embedded systems. However, a great computation load is placed on systems that implement autocorrelation. As seen in Equation 1, the predominant operation in the autocorrelation function is multiplication, which, in the context of a real-time embedded system, is quite time consuming. Furthermore, not all embedded systems have the computational capability to dynamically perform rapid multiplications within a time series. This is the case with the bigger picture heterogeneous multiprocessor system on chip (HMPSoC) under development. The purpose of this embedded system is to analyze an incoming audio signal with a

suite of modes that dictate how to manipulate data. This HMPSoC will contain a ReCOP [7] as one of its only dynamic processing elements. The ReCOP does not implement the functionality to multiply data, and on the chance that it would, sending a stream of data into an important configuration entity would invalidate real-time and reactive system requirements. In consideration of the available processing capabilities, it is clear that an application specific processing body should be implemented on the HMPSoC such that the autocorrelation of an audio signal is efficiently and continuously carried out through a dedicated channel. The form of this solution is an application specific hardware accelerator which implements the core functionality of providing the autocorrelation between a signal and its lagged self. The application specific processor (ASP) will offer additional flexibility for the user of the system, including configuration options such as correlation window size and sample delay before carrying out another correlation. Implementing the COR-ASP will fulfil the technical requirement for the HMPSoC to efficiently evaluate the autocorrelation of an audio signal while also upholding real-time constraints.

Methods

Conceptual Design

The process of successfully designing a hardware accelerator begins with conceptual design. This process involves formulating a physical representation of the operations carried out by the accelerator. In order to complete this task, an object interconnection software was employed which provided diagrams that furthered understanding and clarity.

Black Box

The first step in the conceptual design was deciding on a framework to use for the component. The “Black Box” methodology simplifies complex systems into an abstracted function that describes their inputs and expected outputs at a high level. The COR-ASP will receive a global enable, global reset, and data in from the network on chip (NoC). The COR-ASP will output the autocorrelation of the input signal, providing useful information regarding extracted features of the data (Figure 2).

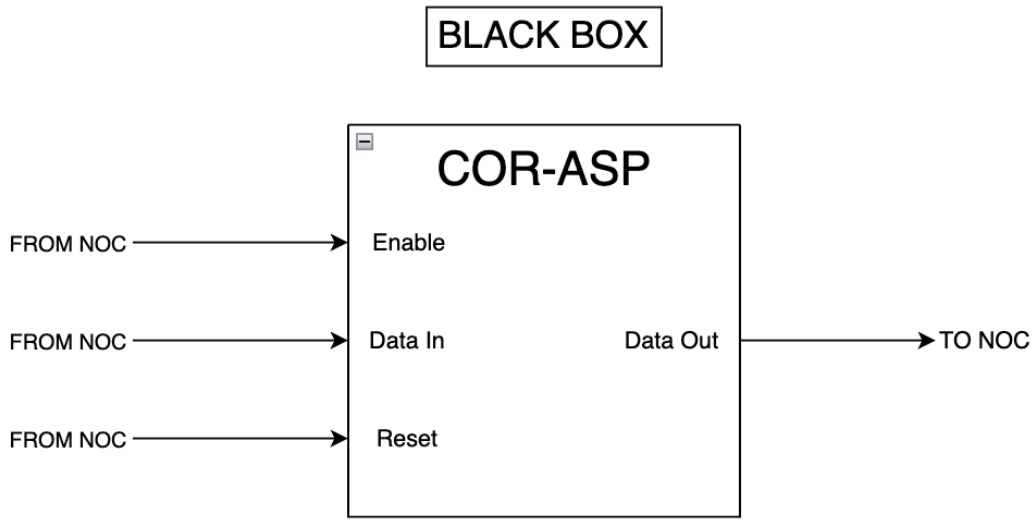

Figure 2: COR-ASP Black Box Diagram

Network on Chip Diagram

As the COR-ASP is designed to be embedded into a HMPSoC NoC, it is important to remain explicit about where the black box inputs and outputs are coming from and going, respectively. As seen in Figure 3, the TDMA-MIN network ties together a collection of system components, accelerators, and processors. Each entity owns a sending and receiving port such that it may communicate with any other entity on the network. The port is 32 bits wide, and data persists on the wire for one clock cycle. Knowing this, it is clear that the COR-ASP should be designed such that it can easily be integrated into the NoC.

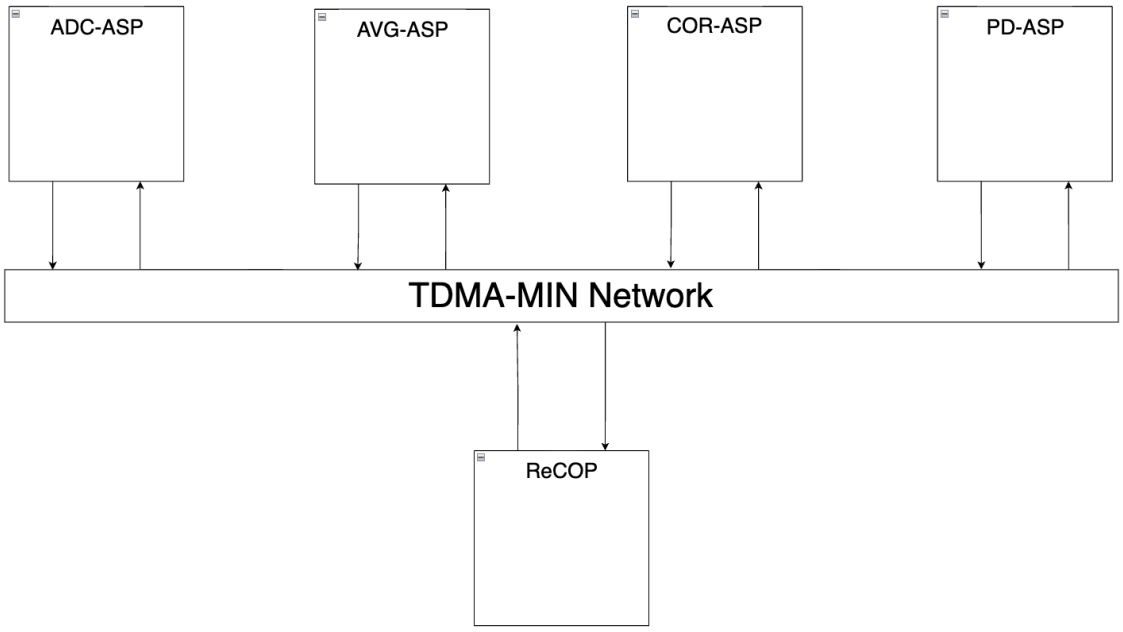

Figure 3: COR-ASP Black Box Diagram

Data Packet

As an entity on the NoC, the COR-ASP must send and receive 32-bit data packets through two ports. The data packet to the COR-ASP must be formatted such that the component can read the type of message it received, potential configuration options, and incoming data. For this reason a packet format was organized and is shown in Packet 1.

[31..28]	[27..24]	[23..20]	[19..18]	[17]	[16]	[15..12]	[7..11]	[6]
1010	Address	N/A	Bit Mode	Enable	Reset	ADC Wait	Corr Win	Passthru

Packet 1: CORE-ASP seven configurable properties

In the scenario that that the packet is not from a configuration entity, the relevant data will be a number from bits [15..0].

Refer to Packet 2 for the format of the CORE-ASP output packet. The reason the data packet extends to 28 bits is due to the nature of the data size. The input data is at maximum 12 bits. The correlation

window is at maximum 32. The bits required to represent the product of two 12-bit numbers is 24. The sum of products is at maximum a 24 bit number multiplied by 2^5, as there are at maximum 32 items in the correlation window. Hence, at its maximum, the output packet will need 29 bits to fully represent the sum. As the input signal does not remain at the maximum value, and lower resolutions are available, it was decided to keep the 4 bits at the front open for the packet ID and only reserve 28 bits for the data.

[31..28]	**[27..0]**
Packet Type ID: 1001	Autocorrelation

Packet 2: CORE-ASP data packet

Datapath Diagram

As the final part of the conceptual design, a complete datapath is depicted in the form of hardware elements such that the process of transferring the design into VHDL code is trivial. Refer to Figure 4 for a depiction of the following functionality.

- The datapath receives information from the NoC
- The packet is decoded into byte segments
- The message type is determined
- If the message type is config
  - Reset indices
  - Load configuration registers
  - Flush correlation
- If the message type is data
  - Load the shift array
  - Increment the number of ADC samples received
- If the message type is neither data nor config
  - Check if the new ADC sample count is sufficient
  - Correlate the contents of the shift array
  - Reset indices
  - Reset new ADC sample count
  - Send autocorrelation over the NoC

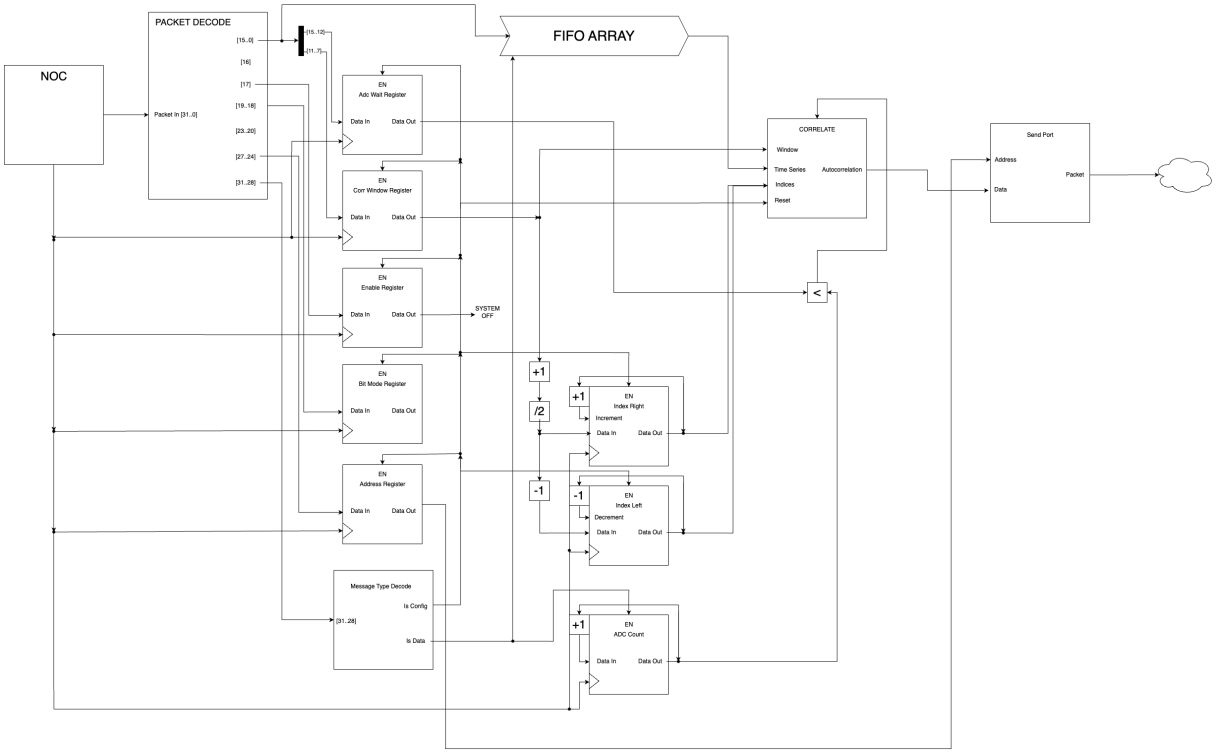

Figure 4: COR-ASP high-level datapath

Mathematical Formulation

Before work is done to implement the autocorrelation function in VHDL, a clear understanding of the underlying mathematics must be demonstrated. First of all, the simplified formula for autocorrelation for an unsigned waveform is provided by Equation 5, where $R[k]$ is the autocorrelation of time series $t(x)$ at lag k and N is the size of the correlation window divided by two.

$$R[k] = \sum_{n=0}^{N-1} t[k+n] * t[k-n-1]$$

Equation 5: Autocorrelation for an unsigned time series t(x)

This equation was implemented in Desmos to provide proof of functionality. Firstly the function was excited with a normal sinusoidal waveform. In Figure 6, the offset sinusoidal waveform $\sin(x) + 2$ was passed through the autocorrelation function. The red line represents the input sinusoid and the green line is the autocorrelation of the signal. As expected, the peaks are highlighted. However, this doesn't provide much for the system, as the input signal is already clean.

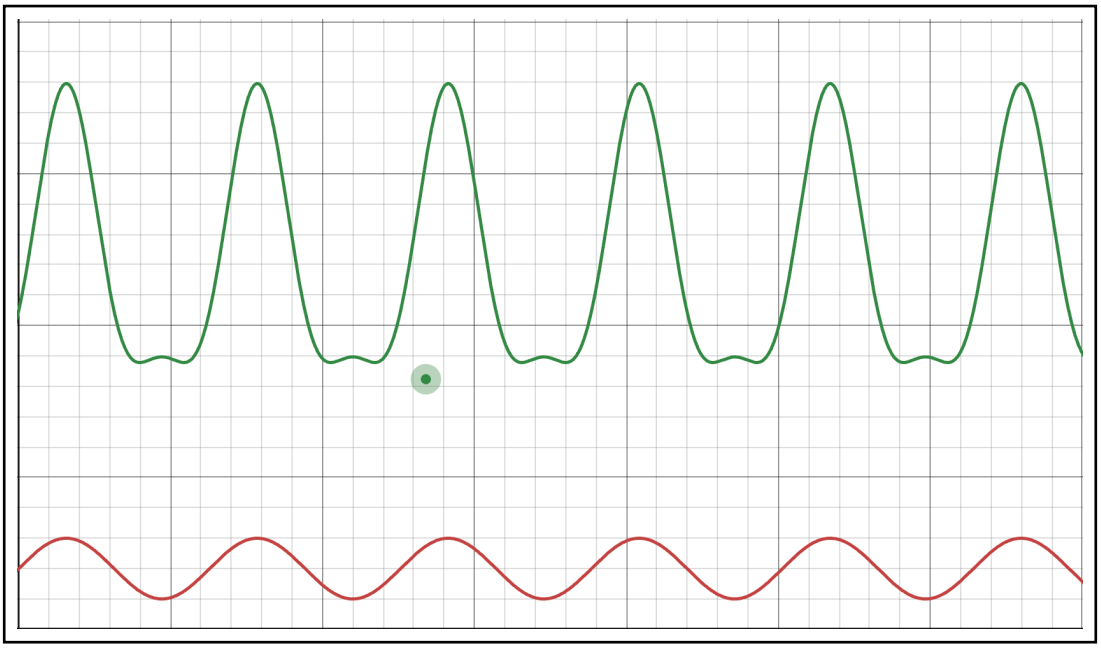

Figure 6: Autocorrelation of a normal sinusoidal waveform

In the next example, a sinusoidal waveform obstructed by a sinusoidal noise is input into the system. Clearly, passing the input signal through a peak detector would cause inaccuracy as local maxima are not characteristic of the waveform's peaks. The autocorrelation function provides a signal where the peaks are straightened out and clearly show the pattern of the waveform (Figure 7).

Figure 7: Autocorrelation of a normal sinusoidal waveform

With sufficient evidence to prove the validity of Equation 5, the next step is to implement the function in VHDL such that it can be synthesized as a hardware accelerator IP.

VHDL

The context of the COR-ASP is to be established as a special purpose processor on a NoC to communicate with controlling bodies and other ASPs. As such, the interface will be defined as follows:

- Input
  - Clock
  - Global Reset
  - Global Enable
  - Network Receive Port
- Output
  - Network Send Port

The next part of the coding is to instantiate the configuration registers. These registers hold important information regarding the current mode of the ASP. In total, there are 6 configurable elements:

`Internal Enable` provides the HMPSoC with the ability to disable the COR-ASP without needing a dedicated enable signal.

`Correlation Window` configures the amount of data points to use when calculating the autocorrelation of a time series.

`ADC Wait` dictates how many new averaged ADC samples to take before performing subsequent correlations.

`Bit Mode` is an unimplemented configuration option that determines the accuracy at which to truncate data.

`Address` configures the sending address for the COR-ASP. That is, to tell the COR-ASP where to send its data to.

`Passthrough` configures the COR-ASP to send received data directly to the sending address, functionally skipping the correlation process. This is used to simplify the configuration process such that there is less of a need to change the target address at startup.

The COR-ASP starts as soon as a configuration message is received. The COR-ASP uses the configuration fields to initialize the configuration registers and the center correlation index. The component then awaits the specified number of ADC samples to populate the shift array. Once all prerequisites are met, the processor begins to compute the autocorrelation of the time series stored in the shift array. A full autocorrelation function executed with a for loop would dramatically decrease the clock frequency limit, especially when working with large correlation window sizes. This is due to the autocorrelation function's $O(n^2)$ complexity placing a heavy multiplicative load on the FPGA. For that reason, the COR-ASP is pipelined; A product pair is calculated at most once every clock cycle and stored in a temporary sum. Once the correlation has concluded, the sum is passed to the send port. The data is sustained for exactly one clock cycle as required by the TDMA-MIN network.

Testing and Verification

Modelsim Programs

In order to verify the proper implementation of the autocorrelator ASP, a comprehensive testing suite is required. For this, a Modelsim testbench was written to display the functionality of the COR-ASP. The testbench instantiates the COR-ASP as a component and uses abstracted TMDA-MIN send

and receive ports to interface the component. Configuration messages are emulated to properly run the processor, and data messages are emulated to populate the shift array (Snippet 1).

```
t_recv_data <= address_constants.message_type_config & "0000" & "0000" & "00" & '1'
& '0' & "0101" & "11111" & "0000000";

wait for 40 ns;
t_recv_data <= address_constants.message_type_average & "000000000000" & x"BEEF";
wait for 40 ns;
t_recv_data <= address_constants.message_type_average & "000000000000" & x"FEED";
```

Snippet 1: Example testbench message packets

The testbench runs as expected and outputs the expected value for the autocorrelation operation. Additionally, the signals are sustained for the correct amount of time and the component is robust to all erroneous messages.

Synthesis Results

The design was synthesized to evaluate the resource usage and timing analysis. As seen in Figure 8 and Figure 9, the COR-ASP is quite efficient considering the complex load it must deal with.

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	116.01 MHz	116.01 MHz	clock	

Figure 8: COR-ASP timing analysis

Fitter Status	Successful - Thu May 23 17:51:11 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard
Revision Name	cor_asp
Top-level Entity Name	cor_asp
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	407 / 32,070 (1 %)
Total registers	607
Total pins	83 / 457 (18 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total RAM Blocks	0 / 397 (0 %)
Total DSP Blocks	1 / 87 (1 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0

Figure 9: COR-ASP resource usage

Future Improvements

One of the most significant downsides of the current autocorrelation ASP implementation is its inaccuracy. The COR-ASP discards the top 16 bits of the result such that is can fit on the TDMA-MIN sending port. By nature, a sum of products of 16 bit numbers is going to be much larger than 16 bits, so ideally 32 bits of memory should be reserved. To do this, a shared memory component may be initialized for use by the COR-ASP and other components. The RAM should be 32 bits wide and have enough lengthwise storage to hold many temporary results. The COR-ASP would directly populate the RAM with new autocorrelations instead of sending the data through the NoC. Then, the COR-ASP would send the next component an address to locate the 32-bit autocorrelation. This method would ensure all accuracy is maintained and no readings are truncated.

Conclusion

Autocorrelation is a measure of a time series significance with respect to a lagged version of itself. Evaluating the autocorrelation of a waveform allows analysts and computers to discover repeating patterns and magnify areas of high energy. The use of autocorrelation is important in signal processing applications for its ability to extract frequencies from noise obstructed waveforms. This process prepares waveforms for peak detection by clearly denoting the local maxima in the signal. While a valuable asset to an embedded signal processing application, the computational load necessary to compute autocorrelation is not easily implemented with general purpose processors. For that reason it is of interest to use hardware acceleration to augment the processing capabilities of a HMPSoC, enabling the embedded system to efficiently carry out autocorrelation.

Appendix

	Bits																					
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	...	0
Data-Audio	1	0	0	0	Dest				Reserved							Ch	16-bit signed integer							
Conf-DP	1	0	0	1	Dest				Next				Mode				Unused							
Conf-ADC	1	0	1	0	Dest				Next				SR		En	Ch	Unused							
Conf-DAC	1	0	1	1	Dest				Reserved				SR		En	Ch	Unused							

Appendix 10: Original TMDA-MIN packet configuration

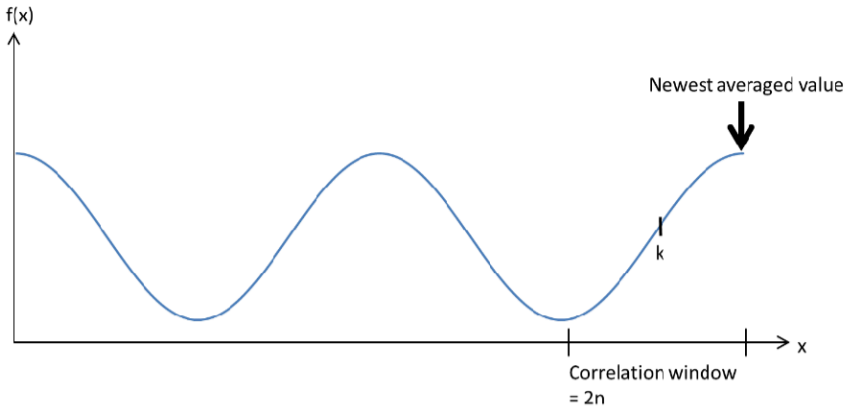

Figure 2. *Diagram showing the window of correlation calculation. The average filtered data is represented by f(x) with the independent variable x delineating each sample. Sample k is the sample around which symmetry is being evaluated. The correlation window size is 2n.*

Appendix 11: Explanation of autocorrelation

Bibliography

[1] J. A. Gubner, *Probability and Random Processes for Electrical and Computer Engineers*. Cambridge University Press, 2006.

[2] J. G. Box G.E.P. and G. Reinsel, "Time Series Analysis; Forecasting and Control. 3rd Edition." Prentice Hall, Englewood Cliff, New Jersey, 1994.

[3] Y. G. Udny, "A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F. R. SPhil. Trans. R. Soc.." 1925.

[4] A. A. Ariyo, A. O. Adewumi, and C. K. Ayo, "Stock Price Prediction Using the ARIMA Model," in *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, 2014, pp. 106–112. doi: 10.1109/UKSim.2014.67.

[5] P. Pal and P. P. Vaidyanathan, "Coprime sampling and the music algorithm," in *2011 Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE)*, 2011, pp. 289–294. doi: 10.1109/DSP-SPE.2011.5739227.

[6] T. H. Pham, S. A. Fahmy, and I. V. McLoughlin, "Low-Power Correlation for IEEE 802.16 OFDM Synchronization on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 8, pp. 1549–1553, 2013, doi: 10.1109/TVLSI.2012.2210917.

[7] Z. Salcic and H. Lorigan, "A Processor Core for Control-dominated Reactive Applications," *University of Auckland, Department of Electrical and Computer Engineering, Embedded Systems Research Group*, 2020.