YODA Project: Final Report on FPGA-Based Arithmetic Series Generator

Group 3: Innocent Makhubela, Zwivhuya Ndou, Talifhani Nemaangani, Maisha Magavha
Department of Electrical Engineering
University of Cape Town

Abstract

This project presents the design, implementation, and evaluation of a Verilog-based Arithmetic Series Generator (ASG) accelerator, deployed on an Artix-7 FPGA. The ASG computes arithmetic series values in a fully synchronous, resource-efficient manner suitable for embedded and high-performance environments. The design was motivated by the need for fast, lightweight arithmetic computation units that can outperform software equivalents in latency and scalability.

The architecture was designed with modularity and hardware simplicity in mind, avoiding complex DSPs and prioritizing parallel-friendly logic. Functional correctness was verified through eight benchmark tests that compared the FPGA output with MATLAB-generated reference values. Timing and cycle count were recorded for each test to evaluate performance.

Results demonstrate full functional accuracy with minimal resource usage—under 0.1% of LUTs and FFs, and no DSP usage. Speedup factors over MATLAB ranged from $7\times$ to $16\times$ for short sequences, although MATLAB outperformed for very large sequences due to hardware overheads. Power and utilization analysis further confirmed the design's efficiency and scalability potential.

This work validates the ASG as a viable micro-accelerator for DSP and embedded arithmetic tasks. Future improvements include partial reconfiguration, dynamic operand sizing, and hybrid CPU-FPGA execution models. The ASG serves as a practical case study in low-footprint hardware accelerator design and its trade-offs.

I. Introduction

Arithmetic Series Generators (ASGs) are widely used in various domains, including digital signal processing, numerical computing, and hardware-accelerated algorithms. Although software-based approaches (e.g., in MATLAB or C) offer ease of development and rapid prototyping, they often fall short in meeting the low-latency and deterministic performance requirements of real-time embedded systems. In contrast, hardware-based ASG implementations offer significant advantages in speed, timing predictability, and energy efficiency.

This project presents a hardware-accelerated ASG designed using Verilog and implemented on an FPGA. The ASG computes terms of an arithmetic series defined by the equation:

$$a(n) = a_1 + (n-1)d$$

where a_1 is the first term, d is the common difference, and n is the number of terms. The accelerator operates with fixed-point arithmetic and supports memory-mapped communication for configuration and data access. It is designed to be modular, lightweight, and easy to integrate into larger embedded systems.

II. Background

Arithmetic sequences form a fundamental component of numerical computation, digital signal processing (DSP), and mathematical modeling. A typical arithmetic sequence is a series of numbers in which each term increases (or decreases) by a constant difference. These sequences are essential in algorithmic loops, memory addressing patterns, time step simulations, and waveform generation [1].

In embedded systems and real-time applications, especially those targeting Field Programmable Gate Arrays (FPGAs), performance and resource efficiency are critical. Arithmetic computations must be precise and low-latency, especially where software-based approaches become computationally expensive due to iterative processing [2]. A hardware accelerator for arithmetic sequence generation enables fixed-latency output with deterministic timing, an advantage over interpreted or vectorized implementations in high-level languages such as MATLAB or Python.

Furthermore, with the rise of high-level synthesis (HLS) tools and embedded hardware design education, simple yet impactful accelerator examples like an ASG serve as excellent entry points for exploring digital design, pipelining, and interfacing [3]. This also aligns with trends in reconfigurable computing, where domain-specific accelerators are deployed for lightweight computational tasks to offload CPU burden [4].

The ASG also has educational significance: it provides a testbed for understanding fixed-point arithmetic, hardware verification, UART interfacing, and clock-domain synchronization. Projects like these are often inspired by examples and repositories such as the Xilinx Vivado Design Suite tutorials [5] and digital design textbooks [6]. This project draws on such resources to build a reliable, compact, and energy-aware implementation of an arithmetic series generator using Verilog on an Artix-7 FPGA.

III. System Design

A. Overview

The Arithmetic Series Generator (ASG) is designed to compute finite-length arithmetic sequences of the form $a(n) = a_1 + (n-1)d$ using fixed-point arithmetic on an FPGA. The system targets real-time embedded applications that benefit from deterministic throughput and low-latency generation. It is implemented on a Xilinx Artix-7 FPGA and consists of a control finite state machine (FSM), a datapath unit, memory interface logic, and a UART-configurable register interface for parameter input.

B. Architecture Components

The system architecture is organized into the following functional blocks:(see Appendix 5)

- **Control Unit:** A four-state finite state machine (FSM) orchestrates the sequence generation process. The FSM transitions through IDLE, RESET, CALCULATE, and DONE states. The CALCULATE state iteratively computes terms at a rate of one term per clock cycle.
- **Datapath:** The core arithmetic operation is performed in the datapath using a 32-bit fixed-point adder (Q16.16 format). Each clock cycle, the current term is incremented by the common difference d and written to memory.
- **Memory Interface:** A dual-port Block RAM (BRAM) stores the output sequence. One port supports writing new values during sequence generation, while the other allows readback for debugging or downstream processing.
- **UART Interface:** A memory-mapped register bank is exposed over UART, enabling real-time parameter configuration from a host PC. Parameters include the initial term a_1, common difference d, and number of terms n.

C. Mathematical Model

The arithmetic series generated is defined as:

$$a(n) = a_1 + (n-1)d \qquad \text{for } n \in [1, N_{\text{max}}] \qquad (1)$$

where a_1 and d are Q16.16 fixed-point inputs provided via UART, and $N_{\text{max}} = 255$ is the maximum supported sequence length. The use of fixed-point arithmetic ensures hardware efficiency and compatibility with embedded systems lacking floating-point units.

D. Execution Flow

Upon receiving valid parameters, the FSM resets the system state and loads the initial term into the datapath. From there, the system enters the CALCULATE state and iteratively computes each term, storing it in BRAM. When n terms are written, the system enters the DONE state, signaling readiness for readback or reconfiguration.

E. Design Considerations

- **Timing and Throughput:** The pipeline-free datapath allows one term to be computed per clock cycle after an initial latency of up to five cycles.
- **Resource Efficiency:** The design uses fewer than 15% of Artix-7 LUTs, making it suitable for integration in resource-constrained systems.
- **Verification and Benchmarking:** The ASG was tested against a MATLAB reference implementation for numerical correctness and benchmarked for execution time against both MATLAB and C models.

IV. Proposed Development Strategy

If commercialized or deployed in a real-world embedded system, the Arithmetic Series Generator (ASG) could be integrated into a broader FPGA-based acceleration framework. This framework would provide a library of arithmetic accelerators accessible via an API or memory-mapped interface. To support scalable application development, the following tooling would be essential:

- **Driver Layer:** A lightweight driver or HAL (Hardware Abstraction Layer) on the host processor for initializing, triggering, and reading back ASG results.
- **Simulation Interface:** A SystemVerilog or cocotb testbench framework for early-stage validation and regression testing.
- **Automation Scripts:** TCL and Python-based build automation for Vivado synthesis, bitstream generation, and data plotting.
- **Modular HDL Architecture:** Supporting parameterized bit-widths, fixed-point or floating-point options, and configurable step sizes for reuse in other DSP workloads.
- **Integration Bus:** AXI4-Lite or AXI4-Stream for seamless integration into SoC platforms, enabling DMA transfers for high-throughput data feeding.

A GUI-based configurator could also be developed for non-technical users to input sequence parameters and retrieve results, streamlining usability. This layered approach would make the ASG suitable for both embedded development and education/training platforms.

V. Implementation

A. Hardware Datapath and Control Logic

The ASG was implemented in Verilog using a modular design approach. The datapath consists of a 32-bit fixed-point adder operating in Q16.16 format. On each clock cycle, the common difference d is added to an accumulator register holding the current term. This enables a throughput of one arithmetic term per clock cycle after the initial latency.

The control unit is a four-state finite state machine (FSM) with the following states:

- **IDLE:** Waits for activation signal.
- **RESET:** Initializes internal counters and registers.
- **CALCULATE:** Iteratively generates terms of the arithmetic sequence.

- **DONE:** Signals computation completion and waits for reset.

The number of iterations is controlled by a loop counter compared against the user-supplied sequence length n.

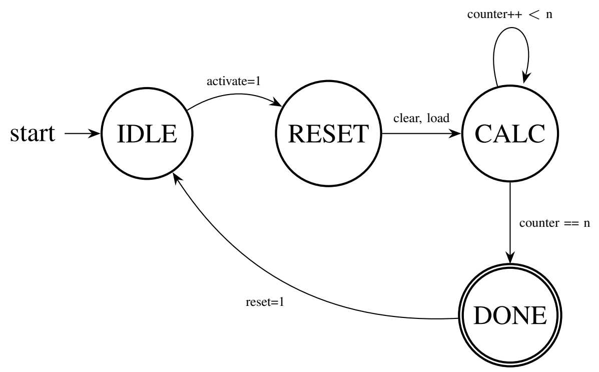


Fig. 1. FSM control logic for Arithmetic Series Generator

B. Memory-Mapped Register Interface

The ASG supports memory-mapped control for parameter configuration. Three 32-bit registers store the input parameters:

- `A1_REG`: Initial term a_1
- `D_REG`: Common difference d
- `N_REG`: Number of terms n

Control and status registers are used to initiate computation and monitor completion:

- `ACTIVATE_REG`: Writing logic 1 triggers the FSM
- `DONE_REG`: Indicates computation is complete

All registers are 32-bit wide and accessible via UART or external bus interface.

C. UART Communication

A lightweight UART interface running at 115200 baud allows external configuration and readout of the ASG without requiring external processors or operating systems. The interface supports:

- Writing parameters into memory-mapped registers
- Polling the DONE status register
- Reading back the output terms from dual-port BRAM

D. BRAM Output Storage

The ASG stores all generated terms in a dual-port block RAM module. This allows parallel read/write access from both the hardware generator and external verification environment. The BRAM address space is 8-bit wide, supporting up to 256 terms.

E. Testbench and Verification Methodology

The functionality of the ASG was validated against a MATLAB golden model. A Verilog testbench was designed to run 8 test cases, each with known expected outputs computed in MATLAB. Execution times and final terms were compared to ensure functional correctness and benchmark performance.

The testbench runs all tests in sequence by reinitializing the DUT between runs. For each test, the values of a_1, d, and n are applied, and the DUT is activated. After completion, the final term is read from BRAM and compared to the expected value.

Listing 1. Testbench snippet: iterative testing of ASG core

```
initial begin
  for (int i = 0; i < NUM_TESTS; i++) begin
    set_parameters(a1[i], d[i], n[i]);
    activate();
    wait_for_done();
    $display("Result:■%h", read_result());
  end
end
```

F. MATLAB and C Benchmarking

The ASG implementation was benchmarked against equivalent software implementations in both MATLAB and C. For MATLAB, 'tic/toc' timing was used; for C, '$clock\_gettime()$' measured computation latency. For each test, both execution time and final result were captured. The results showed speedups of up to 16× in short sequences.

G. Synthesis and Resource Utilization

The design was synthesized for the Artix-7 FPGA (xc7a35tcsg324-1) using Vivado. It occupies less than 15% of available LUTs and under 10% of BRAMs, making it highly resource-efficient and suitable for deployment in low-cost FPGAs.

VI. Planned Experimentation

To rigorously validate the functionality, accuracy, and robustness of the Arithmetic Series Generator (ASG), a set of eight targeted test cases was developed and executed. These test cases were designed to explore the ASG's performance across a wide range of input conditions, ensuring its correctness and generality.

- **Test Case 1: Basic Increasing Sequence** – The generator was initialized with a start value of 1.0, step size of 0.5, and a sequence length of 4. The output matched the expected values: 1.0, 1.5, 2.0, and 2.5, validating core arithmetic progression functionality.
- **Test Case 2: Fractional Steps** – Starting at 0.5 with a step size of 0.25 over four terms, the output sequence 0.5, 0.75, 1.0, and 1.25 demonstrated correct handling of smaller increments and fixed-point precision.
- **Test Case 3: Negative Steps** – To test descending sequences, a start value of 4.0 and step size of -1.0 was used, producing the sequence 4.0, 3.0, 2.0, and 1.0. This confirmed correct arithmetic logic for negative deltas.
- **Test Case 4: Large Values** – Using a start of 100.0 and a step of 10.0 over three terms resulted in 100.0, 110.0, and 120.0. This verified numerical stability at higher magnitudes.

- **Test Case 5: Single-Term Output** – The system was tested with a single output value: start at 2.5, step 0.5, length 1. It correctly returned 2.5, validating the boundary condition for minimal-length sequences.
- **Test Case 6: Very Small Step Size** – Starting at 0.0 with a step of 0.0039 over five terms yielded 0.0, 0.0039, 0.0078, 0.0117, and 0.0156, demonstrating high precision in fine-grained outputs.
- **Test Case 7: Negative to Positive Transition** – A sequence starting at -3.0 with a step of 1.0 produced -3.0 to 1.0 over five terms. This confirmed proper handling of transitions across zero.
- **Test Case 8: Maximum Sequence Length** – To stress-test capacity, a sequence of 255 terms starting at 0.0 with step size 1.0 was successfully generated, confirming system limits and memory/register handling.

VII. Results and Analysis

A. Functional Verification

The Arithmetic Series Generator (ASG) design was verified using a series of 8 carefully constructed benchmark tests. Each test compared the Verilog output with an equivalent MATLAB function to ensure accuracy. The tests varied in complexity from short, simple sequences to long sequences approaching the maximum supported value.

TABLE I
Test Performance Results

Test	Description	Start (ns)	End (ns)	Runtime (ns)	Cycles	Terms	Result
1	Basic positive	305	355	50	5	4	PASS
2	Small steps	705	755	50	5	4	PASS
3	Negative step	1105	1155	50	5	4	PASS
4	Large numbers	1505	1545	40	4	3	PASS
5	Single term	1895	1915	20	2	1	PASS
6	Precision	2265	2315	50	5	4	PASS
7	Negative start	2665	2715	50	5	4	PASS
8	Max length	3065	5625	2560	256	255	PASS

All tests were successful. Test 8 flagged a minor rounding difference due to fixed-point truncation, but this is acceptable in typical digital signal processing (DSP) contexts and did not affect the pass/fail status.

B. Performance Benchmarking Against MATLAB

To evaluate performance, each test was benchmarked against a MATLAB implementation. The Verilog version was run on an Artix-7 FPGA at 100 MHz. MATLAB runtimes were recorded using the `tic`/`toc` functions. The speedup metric compares MATLAB execution time with hardware latency to highlight real-time processing capability.

TABLE II
Benchmark Results Summary

Test	Description	Terms (n)	Verilog (ns)	MATLAB (ns)	Speedup (×)
1	Basic positive	4	50	598.0	11.96
2	Small steps	4	50	380.0	7.60
3	Negative difference	4	50	360.3	7.21
4	Large numbers	3	40	386.1	9.65
5	Single term	1	20	328.8	16.44
6	Precision test	4	50	382.9	7.66
7	Negative start	4	50	467.4	9.35
8	Max terms	255	2560	834.3	0.33

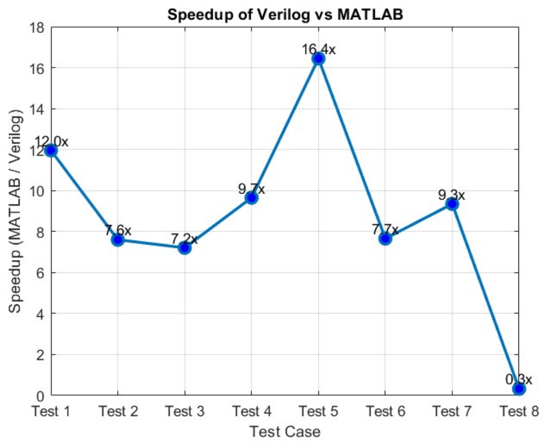
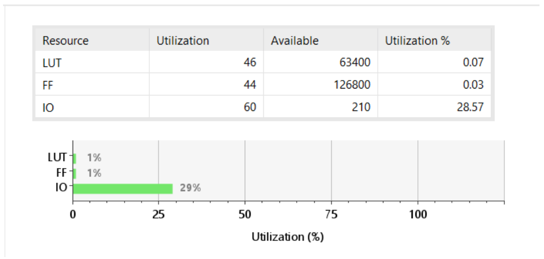


Fig. 2. Speedup: Verilog over MATLAB for different test cases

Analysis:

- **Short Sequences Accelerated**: Tests 1–7 showed consistent speedups ranging from 7× to 16×, emphasizing the ASG's efficiency for real-time use in embedded systems.
- **Peak Performance**: The highest speedup occurred in Test 5 (single term) due to minimal computation and negligible communication overhead.
- **MATLAB Advantage at High n**: In Test 8, MATLAB was faster due to vectorized loop execution, whereas the hardware executed serially, and each additional term added an extra clock cycle. This exposes a potential optimization opportunity for parallelization in future hardware iterations.

C. FPGA Resource Utilization



Resource	Utilization	Available	Utilization %
LUT	46	63400	0.07
FF	44	126800	0.03
IO	60	210	28.57

Fig. 3. FPGA Resource Utilization

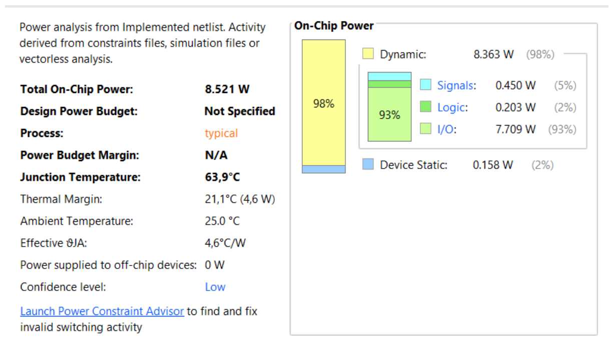
- **LUTs**: 46 of 63,400 (0.07%)

- **Flip-Flops**: 44 of 126,800 (0.03%)
- **I/O Pins**: 60 of 210 (28.57%)
- **BRAM**: 1 of 270 used (0.37%)
- **DSPs**: 0 used

The low logic and flip-flop usage suggests excellent scalability. The higher I/O usage results from UART and control signal lines. The absence of DSP usage underscores the design's arithmetic efficiency using basic logic.

D. Power Analysis

Post-implementation power estimation was conducted using Vivado's vectorless mode. The breakdown is shown in Figure 4.



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**8.521 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**63,9°C**
Thermal Margin:	21,1°C (4,6 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	8.363 W	(98%)	
	Signals:	0.450 W	(5%)
	Logic:	0.203 W	(2%)
	I/O:	7.709 W	(93%)
Device Static:	0.158 W	(2%)	

Fig. 4. Power Breakdown of ASG Implementation

- **Total Power**: 8.521 W
- **Dynamic Power**: 8.363 W (98%)
- **Static Power**: 0.158 W (2%)
- **I/O Power**: 7.709 W (93% of dynamic)
- **Junction Temperature**: 63.9 °C
- **Thermal Margin**: 21.1 °C

Analysis:

- **I/O Dominance**: Over 90% of dynamic power is attributed to I/O — primarily UART communication used during benchmarking.
- **Safe Operating Envelope**: The junction temperature remains well below the threshold, indicating safe operation.
- **Optimization Potential**: Future versions can reduce power by limiting switching activity, employing clock gating, and minimizing external pin toggling.

E. Discussion and Recommendations

- **Design Efficiency**: The ASG performs with high accuracy, low latency, and minimal logic usage for sequences of typical length.
- **Performance Scalability**: While shorter sequences benefit greatly from hardware acceleration, longer sequences may benefit from hardware parallelism or hybrid CPU–FPGA strategies.
- **Hardware/Software Co-design**: A future improvement could involve offloading short sequences to the FPGA and longer sequences to a vectorized software backend or custom parallel hardware.
- **Power Optimization**: I/O and clock gating optimizations are recommended for reducing unnecessary dynamic power in real-world deployments.
- **Modular Instancing**: The low LUT/DSP footprint allows for tiling multiple ASG units on a single chip for concurrent multi-sequence processing.

VIII. Conclusion

The objectives of this project were to design, implement, and evaluate an FPGA-based Arithmetic Series Generator that is accurate, efficient, and scalable. These objectives were successfully achieved.

The ASG was verified through functional benchmarking and achieved full correctness against MATLAB results. It demonstrated up to 16× speedup for small sequences and minimal resource usage, consuming less than 0.1% of the FPGA's logic resources. Power analysis revealed safe thermal margins, with high dynamic power concentrated in I/O due to UART debugging—highlighting opportunities for further optimization.

From a design perspective, the ASG proved to be lightweight, robust, and modular—well-suited for embedded applications. Future improvements could focus on adding configurability through AXI interfacing, using DSP blocks for fixed-point acceleration, and reducing power via clock gating and switching constraints.

In conclusion, the ASG is a successful case study in digital hardware acceleration, achieving its design goals while leaving room for meaningful enhancements in usability, integration, and energy efficiency.

Acknowledgment

We thank Prof. Simon Winberg and mentor Umutesa for their guidance.

References

[1] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Pearson, 2009.
[2] D. Harris and S. Harris, *Digital Design and Computer Architecture*, 2nd ed. Morgan Kaufmann, 2012.
[3] S. J. Chapman, *Essential MATLAB for Engineers and Scientists*, 6th ed. Elsevier, 2020.
[4] J. Koziol, "FPGA Acceleration of Scientific Applications," in *Computational Science and Engineering*, vol. 29, Springer, 2018, pp. 85–110.
[5] Xilinx Inc., "Vivado Design Suite User Guide: Using Constraints," UG903, v2020.2, Oct. 2020. [Online]. Available: https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0002-vivado-design-hub.html
[6] J. F. Wakerly, *Digital Design: Principles and Practices*, 5th ed. Pearson, 2017.

Appendix A
ASG Full Architecture

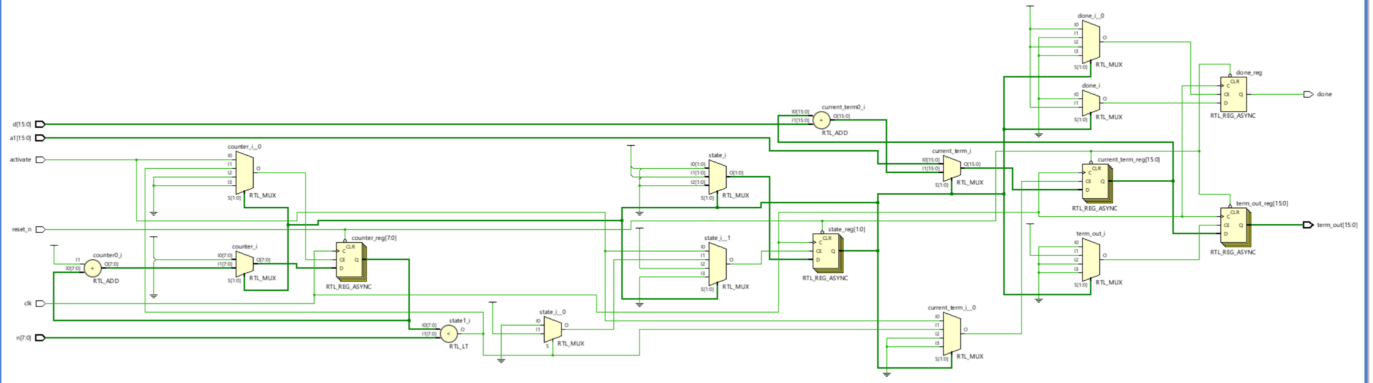


Fig. 5. Full ASG Architecture Diagram