# Design, Layered Verification and FPGA Validation of a Register-Programmed AXI4 DMA Controller for Streaming Image Pipelines

Harshada Dalvi[1], Pravin Bhangare[2], Vaishnavi Bauchkar[3], Ameya Pethe[4], Anuradha Joshi[5]
[1,2,3] *Dept. of Electronics and Telecommunication Engineering Vidyalankar Institute of Technology Wadala, Mumbai, India*
[4,5] *Assistant Professor, Dept. of Electronics and Computer Science Vidyalankar Institute of Technology Wadala, Mumbai, India*

***Abstract*—Streaming image hardware on a Zynq-class system-on-chip relies on a DMA engine to carry pixels between DDR memory and the filter logic, so a defect in that engine shows up as a damaged picture long before anyone suspects the filter. This paper describes a small register-programmed DMA core with separate memory-to-stream and stream-to-memory channels, a 32-bit AXI4 memory side, and an 8-bit AXI4-Stream side that moves one pixel per beat. We pair the design with a layered SystemVerilog testbench built around a reference-memory scoreboard, five passive protocol monitors, four interface timing profiles and a 46-bin coverage list. Under Vivado xsim the regression ran 295 test items and 1,622,060 checks with no errors and every coverage bin hit. A 65,536-byte frame went through the loop with no mismatch in 65,645 cycles. A Verilator run reached the same verdict, and each of nine bugs planted in the RTL made the regression fail. The core was then integrated with a Gaussian filter on a Zynq-7000 device, where a 256 × 256 8-bit frame (65,536 bytes) was streamed through the filter and written back to DDR in 1.245 ms, about 52.6 MB/s end to end. The simulation figures cover the core alone; the board result was judged by inspection and has not been compared pixel by pixel with a software reference.**

***Index Terms*—DMA controller, AXI4-Stream, SoC design, SystemVerilog verification, functional coverage, fault injection, Gaussian filter, Zynq-7000**

## I. INTRODUCTION

A frame buffer sitting in DDR is of no use to a hardware filter until something turns it into a stream. On a Zynq device that job normally goes to a DMA engine: the processor writes a start address and a length, and the engine then reads memory in bursts, hands the pixels to the filter one per clock, and later accepts the filtered stream and writes it back. When the engine misplaces a byte, the visible symptom is a shifted or speckled image, and it is rarely obvious whether the filter or the data path is at fault.

Vendor DMA blocks solve the problem in general, but they are large, and a student team cannot read or change their internals. For a fixed-function pipeline with one stream width and one image size, a compact core written from scratch is a reasonable alternative, as long as it is checked far more thoroughly than a handful of image runs would allow. Unaligned start addresses, bursts that land on a 4 KB boundary, packets that end early and bus errors seldom occur in normal traffic, which is precisely why they need deliberate stimulus.

The work reported here comes from a combined SoC and ASIC design course and has two parts. The first part is the design of the controller. The second is the verification environment and the evidence gathered with it, in simulation and on a Zynq-7000 board running a Gaussian filter. Section II gives background, Section III describes the controller, Section IV the testbench, Section V the simulation and board results, and Sections VI and VII cover limitations and conclusions.

## II. BACKGROUND AND RELATED WORK

The behaviour expected of the bus interfaces is defined by the AXI4 and AXI4-Stream specifications [1], [2]. Two of their rules shape this design more than any others: a burst must not cross a 4 KB address boundary, and a source must keep its payload steady until the receiver signals ready. Commercial AXI DMA IP [3] offers direct-register and scatter-gather operation together with interrupts, and is the usual choice on Zynq parts [4].

Line-oriented streaming is common in FPGA vision and remote-sensing accelerators. Coca and Datcu [5] feed data line by line into an accelerator on a Zynq UltraScale+ device and report 81 ns per pixel. Madaoui et al. [6] place a small neural network on the Zynq-

7020 that the PYNQ-Z2 uses, and report that it takes up 17% of the LUTs, which suggests that this device has room left for custom data movement logic. Studies of this kind focus on the compute stage and say little about how the surrounding data path was verified.

For verification we follow the layered, coverage-driven style described in [7] and [8], written in SystemVerilog [9] but without a UVM library, so that the same sources run on Vivado xsim and on Verilator [10]. To judge how sensitive the testbench is, we also used fault injection, in the spirit of mutation testing [11].

## III. CONTROLLER DESIGN

### *A. Structure and Parameters*

The controller, *dma_core*, works in direct-register mode and has no scatter-gather support. An AXI4-Lite slave exposes one register file, a register block holds the state of each channel, and the two engines, *dma_mm2s* and *dma_s2mm*, each sit behind a first-word-fall-through FIFO (Fig. 1). A two-flop synchroniser conditions the active-low reset, and everything runs from a single clock.

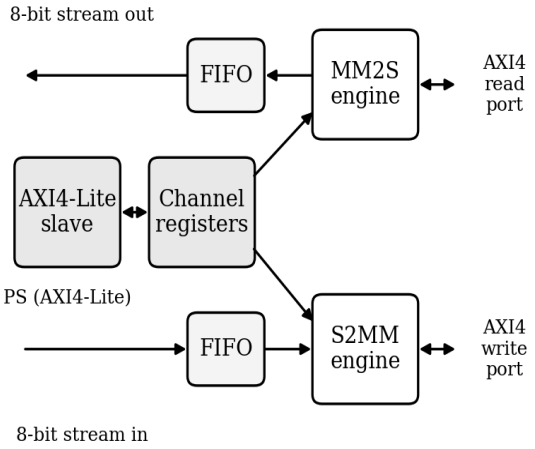


Fig. 1. Block structure of the DMA core.

The memory side is 32 bits wide, the stream side 8 bits wide, bursts are limited to 16 beats, each FIFO holds 64 words, and a transfer length is 23 bits, so one transfer can cover up to 8,388,607 bytes. The FIFO depth must be a power of two and not less than the burst limit; an elaboration-time check enforces this.

### *B. Register Interface*

Table I lists the registers. Software starts the write channel first, so that it is ready when pixels begin to arrive, and then the read channel; writing a non-zero length with the run bit set launches a transfer. Completion is found by polling the IOC bit in the status register, since the core has no interrupt output. An abort is requested by writing 0x5 to the control register and waiting for the reset bit to clear and the halted bit to appear.

TABLE I. REGISTER MAP OF THE CORE

| Offset | Register | Role |
|---|---|---|
| 0x00 | MM2S_CR | Run bit, abort request |
| 0x04 | MM2S_SR | Halted, idle, error and IOC flags |
| 0x18 | MM2S_SA | Source byte address |
| 0x28 | MM2S_LEN | Length; a write starts the transfer |
| 0x30 | S2MM_CR | Run bit, abort request |
| 0x34 | S2MM_SR | Halted, idle, error and IOC flags |
| 0x48 | S2MM_DA | Destination byte address |
| 0x58 | S2MM_LEN | Length limit; holds bytes received afterwards |

### *C. Read Channel*

The read engine rounds the start address down to a 32-bit boundary and counts the unaligned leading bytes in its word total. For every burst it takes the smallest of three quantities: the 16-beat limit, the words still to be fetched, and the words left before the next 4 KB boundary, so the boundary rule is respected by construction rather than by checking.

Overflow is prevented with a reservation counter. Whenever a read is issued, space for it is reserved in the FIFO, and a new burst is only issued if all of its beats will fit. Returning data therefore always has a place to go, which is why the ready signal of the read-data channel can be held high. An unpacking stage then pulls bytes out of each word starting at the right lane and marks the last byte of the transfer with TLAST. On abort the engine stops issuing reads, waits for every outstanding beat to come back, empties the FIFO and reports completion.

### *D. Write Channel*

Incoming bytes are accepted while the channel is active and the FIFO has room. A packing stage gathers them into 32-bit words together with a byte mask, so a partly filled first or last word only modifies the bytes that belong to the transfer. A word enters the FIFO when it is complete or when the final byte arrives, whether that is signalled by TLAST or by reaching the programmed length.

A counter tracks words that are ready but not yet covered by an address, and a burst is issued once a full one is available, or, after the packet has ended, for whatever remains. Up to two burst lengths wait between the address and data channels so that the data phase follows the address phase. A second counter follows outstanding write responses, and the channel declares completion only when the packet has ended, nothing is left in the FIFO and every response has been received. After completion the length register returns the number of bytes that were actually accepted.

*E. Behaviour Worth Noting*

A few properties are intentional or follow from the design and should be known by the integrator. Writes to unmapped addresses are ignored without an error response. The slave-error and decode-error flags stay set until the next transfer begins, and clearing IOC and Err does not clear them. Read-side TVALID is gated by the abort state, so during an abort it can drop without a handshake, which strict AXI4-Stream usage would not normally allow. Finally, the core carries no TKEEP or TSTRB support: every stream byte is taken as valid .

*F. Notes for an ASIC Flow*

Because the core is synchronous, uses one clock and synchronises its reset, it carries over to a standard-cell flow without clock-domain work. The FIFOs are the one part that would change: the FPGA version infers distributed RAM, whereas a cell-based implementation would use a register array or a small memory macro, and its area would have to be re-examined. We did not run synthesis for a standard-cell library, so no area, timing or power figures are claimed for the ASIC target.

## IV. VERIFICATION ENVIRONMENT

*A. Layers and Models*

The testbench has four layers (Fig. 2). Tests at the top only describe transfers. The functional layer creates transaction objects, predicts the expected result with a scoreboard that owns a separate copy of memory, and counts coverage. The command layer contains a driver that programs the registers, polls, aborts and resets, together with five monitors on the AXI4-Lite, read, write, stream-out and stream-in interfaces. At the bottom, the signal layer provides one interface, models of DDR for reading and writing on a 4 MB byte array, and a stream sink and a stage that stands in for the filter.

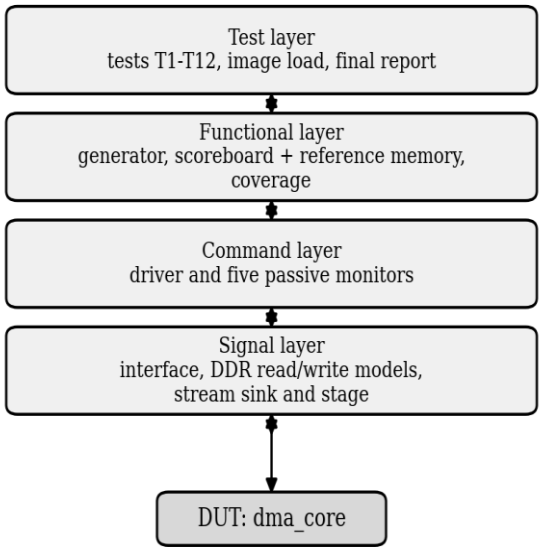


Fig. 2. Layers of the testbench around the DUT.

The read model returns a slave error inside a 4 KB window at 0x0030_0000 and a decode error outside memory. The write model applies the byte strobes to memory and answers each burst with one response.

*B. Stimulus and Timing*

Three modes are used. In loop mode the write channel is armed first, then the read channel with the same length, and the stream is passed through the throttled stage; this is the real application flow. In the other two modes one channel runs alone against a random sink or a random source, the latter with gaps, an early TLAST or surplus bytes. Interface timing follows four profiles (Table II), and in the random profile each parameter is redrawn for every transfer.

TABLE II. INTERFACE TIMING PROFILES

| Profile | Behaviour |
|---|---|
| fast | All ready signals high, no latency |
| slow | Address ready 30%, read gaps 40%, write ready 50%, sink ready 35%, source gaps 40% |
| random | Ready 10-100%, latency 0-16 cycles, stalls 0-60%, redrawn per transfer |
| special | Sink ready 3%; source gaps 85% with write ready 10%; address ready 5% with response delay 20-40 cycles |

*C. Checking*

The scoreboard compares every stream byte on the read channel with the reference memory at the matching address, requires the byte count to equal the programmed length and expects exactly one TLAST, on the last byte. It also confirms that the number of read words equals ceil((offset + length)/4). On the write channel it checks the address and the value of every byte derived from the strobes, the number of address, data and response beats, the position of WLAST, and that no strobe vector is zero. After each transfer the touched region plus a 64-byte guard on both sides is compared with the reference memory, and the whole 4 MB is compared at the end.

Registers are read back after completion and after clearing, and after an abort the control register must read zero with the halted flag set. The monitors enforce burst size, burst type, length, alignment, the 4 KB rule and the valid-until-ready rule on every interface, with the stability check switched off during aborts and resets. A separate comparison at the top level checks the image written to DDR against the input file without involving the scoreboard. A per-transfer timeout of 400,000 cycles and a global watchdog guard against hangs.

*D. Tests and Coverage*

Twelve tests cover register behaviour and zero-length transfers, the full image at two timing profiles, 29 lengths from 1 to 8,191 bytes in all three modes, all sixteen source and destination offset pairs with bursts

around 4 KB boundaries, back-pressure profiles, short and long input streams, a second length write while busy, error responses, aborts at four delays, hardware reset in the middle of a 6,000-byte transfer, FIFO stress, and a constrained-random block with 60 transfers and 12 random aborts.

Coverage is counted with plain counters instead of covergroups. The 46 required bins span test kinds and modes, MM2S and S2MM length classes, source and destination offsets 0 to 3, 4 KB crossings on both channels, three stream-length cases, a length write during a busy transfer, four error combinations and four abort combinations. The regression is accepted only if all checkers are silent, every bin is hit, the image shows no mismatch, nothing times out, and each planted fault is caught.

## V. RESULTS

### *A. Regression in Vivado xsim*

Vivado xsim 2024.2 is the simulator of record. Table III summarises the run, which used the project's own image file. The log was also parsed to confirm that all twelve section banners appear in order, that transaction numbers 0 to 294 are contiguous, that counts per transfer kind equal the coverage counters, and that no ERROR or FATAL line and no checker message is present. The log does contain tool warnings: five testbench variables that lack an explicit automatic or static qualifier, protocol instances of the block design that could not be loaded because the testbench does not instantiate it, a locked-IP notice for the packaged DMA core, and objects that xsim cannot trace in the wave window. None of them comes from a checker. A clean log shows only that no checker fired during this run; the fault-injection study below addresses whether the checkers are able to fire.

TABLE III. SUMMARY OF THE XSIM REGRESSION

| Item | Result |
|---|---|
| Test items | 295 (289 transfers) |
| Checks executed | 1,622,060 |
| Errors | 0 |
| Coverage bins | 46 of 46 |
| Image test | 65,536 bytes, 0 mismatches |
| Simulated time | 6.32 ms at 100 MHz |

### *B. Throughput in Simulation*

In simulation with all ready signals high, the 65,536-byte image completed in 65,645 cycles including register programming, polling and clean-up. That is 109 cycles, about 0.17%, above the ideal one byte per clock, or 100 MB/s at 100 MHz. Under the slow profile the same frame needed 101,437 cycles.

### *C. Second Simulator*

The same sources ran under Verilator 5.4x with the same outcome: 295 items, 289 clean transfers, no errors and 46 of 46 bins. The number of checks (1,521,061) and the simulated time (5.65 ms) differ from the xsim values because the random tests draw different sequences in each tool; the directed tests, the number of transactions and the coverage result are identical.

### *D. Fault Injection*

Nine defects were inserted one at a time into a copy of the RTL and the full regression was repeated. Every one caused a failure (Table IV). Failures came from data, protocol, register, error-handling, address and liveness checks, so detection did not depend on a single mechanism. Nine hand-picked faults is a small sample, and the result should be read as evidence of sensitivity, not as a mutation score.

TABLE IV. INJECTED FAULTS AND HOW THEY WERE CAUGHT

| Injected fault | First failure reported |
|---|---|
| MM2S data bit flipped on one beat | Data mismatch at byte 65,531 |
| S2MM strobes not accumulated | Wrote 16,384 bytes, accepted 65,536 |
| MM2S 4 KB limit off by 4 bytes | AR burst crosses 4 KB |
| MM2S TLAST one beat early | Timeout |
| S2MM LEN readback off by one | LEN 65,537 instead of 65,536 |
| IOC write-1-to-clear disabled | IOC not cleared |
| S2MM ignores write-response errors | Err/SLVERR bits missing |
| S2MM AWADDR wrong for 3-beat bursts | Write address mismatch |
| MM2S wrong length for 16-beat bursts | Timeout, image mismatch |

No functional defect was found in the RTL. One tooling remark: Verilator 5.020 from the Debian package stops on a combinational convergence error with this testbench, while version 5.30 or newer and Vivado xsim run it correctly.

### *E. System Integration and Waveform*

Fig. 3 shows the Vivado block design in which the core is integrated on a Zynq-7000 device. The processing system programs DMA_Controller_1 through an AXI SmartConnect that carries the AXI4-Lite register interface from its general-purpose master port. The read and write master ports of the controller reach DDR through a second SmartConnect and the high-performance slave port of the processing system. On the stream side, the memory-to-stream channel supplies pixels to the Gaussian filter (Gaussian_filter_0), and the filtered stream passes through an AXI4-Stream data FIFO to the stream-to-memory channel. A processor system reset block and a constant source provide the reset and static control signals.

Fig. 4 shows the top-level waveform of the layered testbench (tb_dma_layered) from the default 1,000 ns run that Vivado performs before the full regression is started. The clock has a 10 ns period, which corresponds to the 100 MHz used for the simulation figures in this paper, and the gate input is held high. The constants MAX_I…, IMG_F… and N_RA… read 0x00040000, 0x00000002 and 0x0000003C; the last equals 60, the number of constrained-random transfers in the log. The output word out_m… and the flag img_f…lback stay at zero, as expected, because the log announces the first register test (transaction 0) only at 215 ns and no image has moved in this window. The testbench instantiates the core sources directly, so the block design of Fig. 3 is not itself simulated (Section VI).

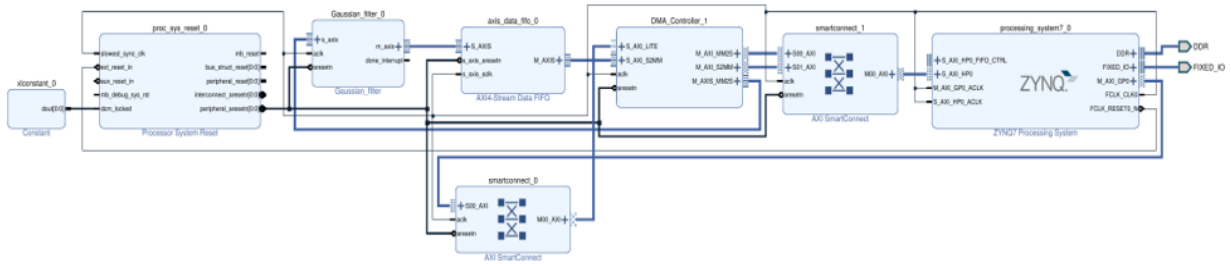


Fig. 3. Vivado block design of the Zynq-based streaming system containing the DMA controller, Gaussian filter, AXI4-Stream FIFO, AXI SmartConnect blocks and the processing system.

Fig. 4.

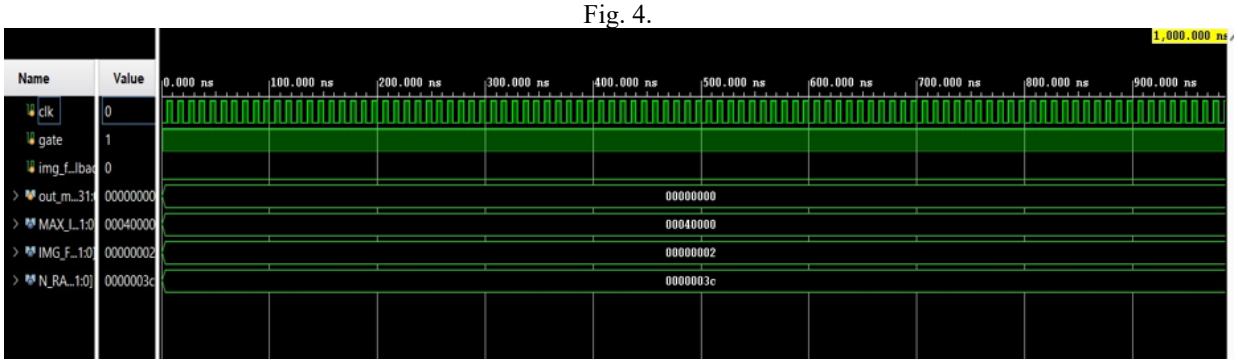


Fig. 5. Testbench top-level waveform over the first 1,000 ns of the xsim run (10 ns clock).

*F. Hardware Validation on the Zynq Device*

The design of Fig. 3 was also run on a Zynq-7000 board with a host script on the processing system. A 256 × 256 image was converted to 8-bit grayscale, which gives 65,536 bytes, and placed in a DDR buffer at 0x18060000; a second buffer of the same size at 0x18070000 received the result. The script reset both channels, programmed the stream-to-memory channel first and the memory-to-stream channel second, as described in Section III, using the register base 0x40000000 and the offsets of Table I, and then waited for completion. Table V lists the outcome.

TABLE V. SUMMARY OF THE HARDWARE RUN

| Item | Result |
|---|---|
| Image | 256 × 256, 8-bit grayscale |
| Transfer size | 65,536 bytes each way |
| Input buffer | 0x18060000 |
| Output buffer | 0x18070000 |
| Register base | 0x40000000 |
| Final MM2S / S2MM status | 0x00001002 / 0x00001002 |
| Transfer time | 1.245 ms (software timed) |
| Average rate | 52.6 MB/s |
| Output pixels | min 0, max 241, mean 59.82 |

Both channels ended with the same status value, 0x00001002. Under the Xilinx AXI DMA bit layout that the register offsets of the core mirror, this value corresponds to the idle and completion flags being set with no halted or error flag asserted.

The measured time of 1.245 ms gives an average of 52.6 MB/s over the 65,536 bytes. In simulation the same frame took 65,645 cycles, or 656 µs at 100 MHz, so the board run was about 1.9 times slower. The two figures are not like for like. The board time is measured in software and therefore also contains the cost of register accesses from the processor, and the data pass through the filter, the stream FIFO, the interconnect and the DDR controller, none of which is in the simulated loop. These contributions were not separated, and the fabric clock frequency was not recorded in the log, so the 100 MHz used for the conversion is an assumption.




Fig. 6. Input image (RGB, left) and the Gaussian filter output read back from DDR after the DMA transfer (grayscale, right).

Fig. 5 compares the input with the filter output. The output is a smooth grey-level version of the scene with the same layout, and it shows none of the shifts, tearing or speckle that a misplaced byte or a wrong address would cause. Its pixel values span 0 to 241 with a mean of 59.82, which fits the large dark background of the picture. This check is visual and statistical only: the output was not compared pixel by pixel with a software Gaussian reference, so exact filter correctness is not claimed.

## VI. LIMITATIONS

The simulation evidence covers the core at its default configuration. The filter, interconnect, stream FIFO and processing system are not modelled in the regression, and the testbench compiles the core sources directly, so the block design is not simulated; Vivado also reports the packaged DMA core in the block design as locked, which should be resolved before the project is reused. Other burst lengths, FIFO depths and data widths were not run. The random tests are not seeded from a plusargument, which makes a failing random case hard to reproduce across tools. Coverage is counter-based and code coverage was not measured, so some RTL paths may never have been exercised. Only one AXI response type per channel per transfer is tested, and the RTL has no assertions. The hardware run covers one 256 × 256 frame. Its output was judged by inspection and summary statistics rather than against a software reference, its time was measured in software, and the fabric clock was not recorded, so the 52.6 MB/s figure describes this system end to end and not the core alone.

## VII. CONCLUSION

We designed a compact register-mode AXI4 DMA controller, checked it with a layered SystemVerilog environment, and ran it with a Gaussian filter on a Zynq-7000 device. Across 20 planned features, 12 tests, 289 transfers, about 1.6 million checks and 46 coverage bins, the xsim regression finished clean, Verilator agreed, and all nine planted faults were detected. In simulation the controller moves about one byte per clock with 0.17% overhead on a 64 KB frame. On the board, a 65,536-byte frame was filtered and returned to DDR in 1.245 ms, about 52.6 MB/s end to end, and the output image shows no sign of data-path corruption.

Planned next steps are a pixel-exact comparison of the board output with a software Gaussian model, timing with a hardware cycle counter to separate DMA, filter and software overhead, a system-level simulation of the block design, verification of non-default parameters, seeded randomisation with code coverage and SVA assertions, an interrupt output, and a standard-cell synthesis run to put numbers on the ASIC side.

## ACKNOWLEDGMENTS

The authors thank the Department of Electronics and Telecommunication Engineering, Vidyalankar Institute of Technology, for the laboratory facilities used in this work.

## REFERENCES

[1] ARM Ltd., *AMBA AXI and ACE Protocol Specification*, ARM IHI 0022.
[2] ARM Ltd., *AMBA 4 AXI4-Stream Protocol Specification*, ARM IHI 0051.
[3] AMD-Xilinx, “AXI DMA LogiCORE IP Product Guide,” PG021.
[4] AMD-Xilinx, “Zynq-7000 SoC Technical Reference Manual,” UG585.
[5] M. Coca and M. Datcu, “FPGA accelerator for meta-recognition anomaly detection: Case of burned area detection,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 16, pp. 5247–5259, 2023.
[6] L. Madaoui, M. Kedir-Talha, and O. Kerdjid, “An FPGA-based embedded system for terrain recognition using inertial measurement units,” in *Proc. 8th Int. Conf. Image Signal Process. Appl. (ISPA)*, 2024.
[7] J. Bergeron, *Writing Testbenches Using SystemVerilog*. New York, NY, USA: Springer, 2006.
[8] C. Spear and G. Tumbush, *SystemVerilog for Verification*, 3rd ed. New York, NY, USA: Springer, 2012.
[9] *IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language*, IEEE Std 1800-2017, 2018.
[10] W. Snyder *et al.*, “Verilator.” [Online]. Available: https://www.veripool.org/verilator
[11] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, “Hints on test data selection: Help for the practicing programmer,” *Computer*, vol. 11, no. 4, pp. 34–41, Apr. 1978.