Investigating BRAM Data Persistence and Security Implications on FPGAs

Aryaman Ghura
Faculty Advisor: Professor Sandip Kundu

Abstract

This research investigates a critical security vulnerability in Xilinx ZCU104 FPGA boards related to Block RAM (BRAM) persistence. We demonstrate that sensitive data processed by one user's kernel remains accessible in BRAM after the kernel execution terminates, enabling subsequent users to extract this information through direct memory access. Our findings reveal that current isolation mechanisms fail to properly clear BRAM contents between different users' applications, creating a significant side-channel vulnerability. This work presents a proof-of-concept attack methodology and proposes mitigation strategies to enhance security in FPGA-based computing environments.

Background

Field Programmable Gate Arrays (FPGAs) are increasingly deployed in multi-tenant cloud environments due to their reconfigurability and acceleration capabilities. The Xilinx ZCU104 platform, featuring Zynq UltraScale+ architecture illustrated in Figure 1, is widely used in research and production settings. Block RAM (BRAM) provides on-chip memory resources that kernels utilize for high-performance data processing. Current security models assume that once a kernel terminates execution, its data is no longer accessible to subsequent users. However, this assumption fails to account for the physical persistence of memory states in BRAM resources, creating a potential attack vector in shared environments where users sequentially access the same FPGA resources.

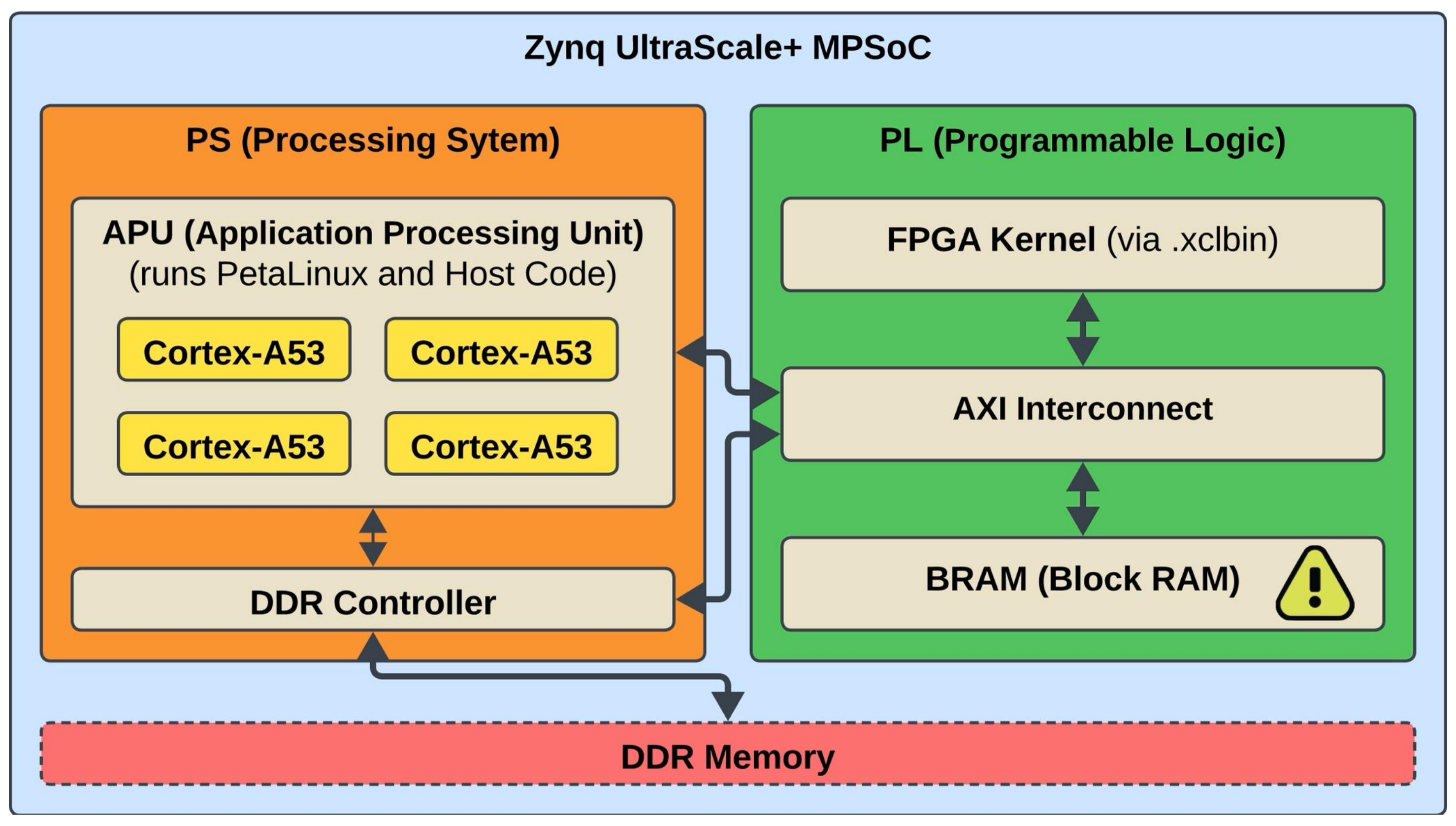


Figure 1: Zynq UltraScale+ MPSoC Architecture

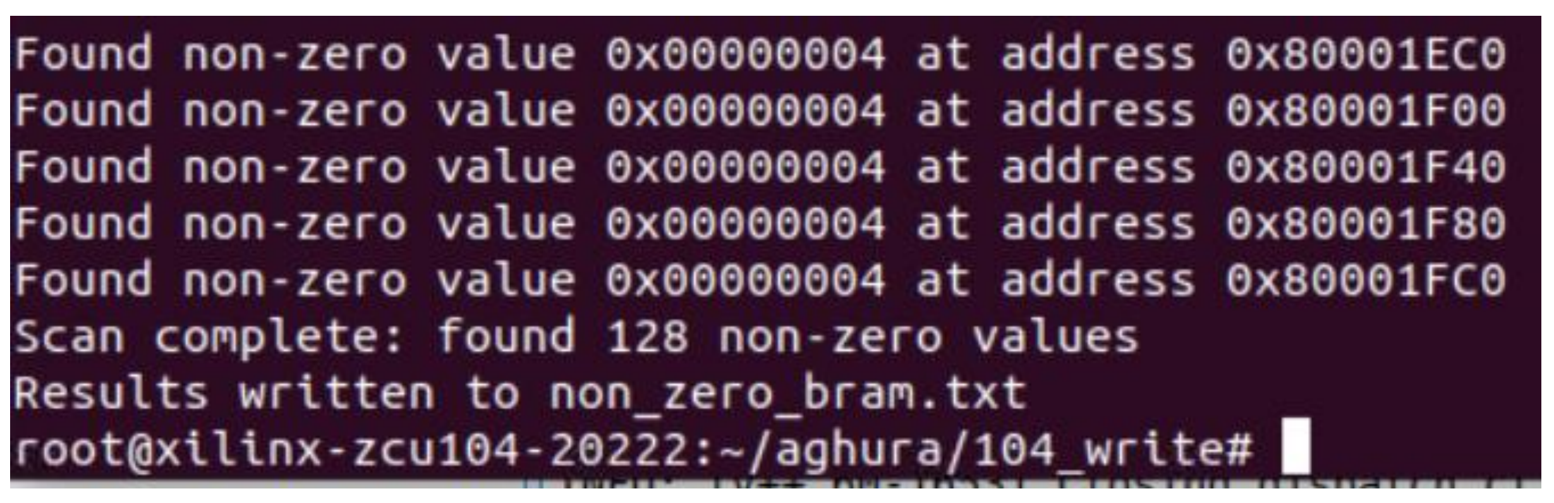
Research Questions

1. Is the sensitive data being processed by the kernel accessible in the BRAM after kernel execution terminates?
2. Can a malicious user extract meaningful information from BRAM persistent data?

Methodology

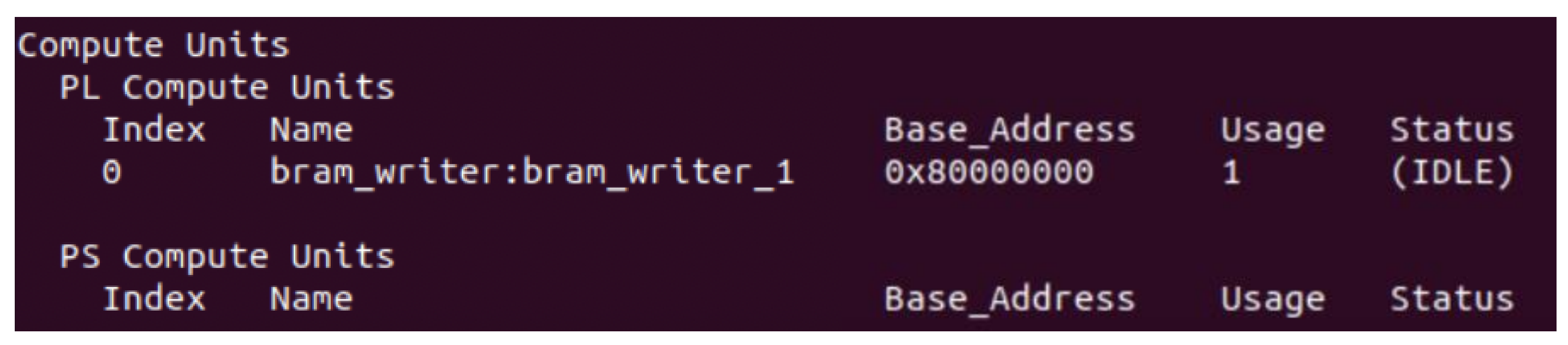
The following steps demonstrate our proposed attack vector:

1. **Environment Setup:** We created a custom PetaLinux image and booted the ZCU104 board via JTAG. This customization was necessary because the default PetaLinux image restricts the Processing System from driving the Programmable Logic.
2. **Memory Zeroing Phase:** We developed a custom utility (zero_bram.c) that accesses BRAM directly through /dev/mem and writes zeros to all accessible BRAM locations, establishing a clean baseline state. Another custom utility (read_bram.c) is implemented to scan the BRAM for any remaining non-zero values and account for them, as shown in Figure 2.



```
Found non-zero value 0x00000004 at address 0x80001EC0
Found non-zero value 0x00000004 at address 0x80001F00
Found non-zero value 0x00000004 at address 0x80001F40
Found non-zero value 0x00000004 at address 0x80001F80
Found non-zero value 0x00000004 at address 0x80001FC0
Scan complete: found 128 non-zero values
Results written to non_zero_bram.txt
root@xilinx-zcu104-20222:~/aghura/104_write# █
```

Figure 2: Reading BRAM after executing the memory zeroing phase. Every 0x40 bit found to retain a fixed value of 0x04.



```
Compute Units
  PL Compute Units
    Index    Name                          Base_Address    Usage   Status
    0        bram_writer:bram_writer_1     0x80000000      1       (IDLE)

  PS Compute Units
    Index    Name                          Base_Address    Usage   Status
```

Figure 3: PL unit configured with the “bram_writer” kernel

3. **Victim Execution Phase:** A standard Xilinx vector addition kernel (bram_writer.xclbin) was executed using OpenCL as shown in Figure 3. This kernel performs vector addition on two input vectors with values 40 and 2, producing an output of 42. The kernel utilizes BRAM for intermediary storage through local memory buffers.
4. **Data Extraction Phase:** After the victim kernel completes execution, we run read_bram.c again to scan the previously zeroed BRAM regions for changed values, identifying and recording data remnants left by the victim's computation.

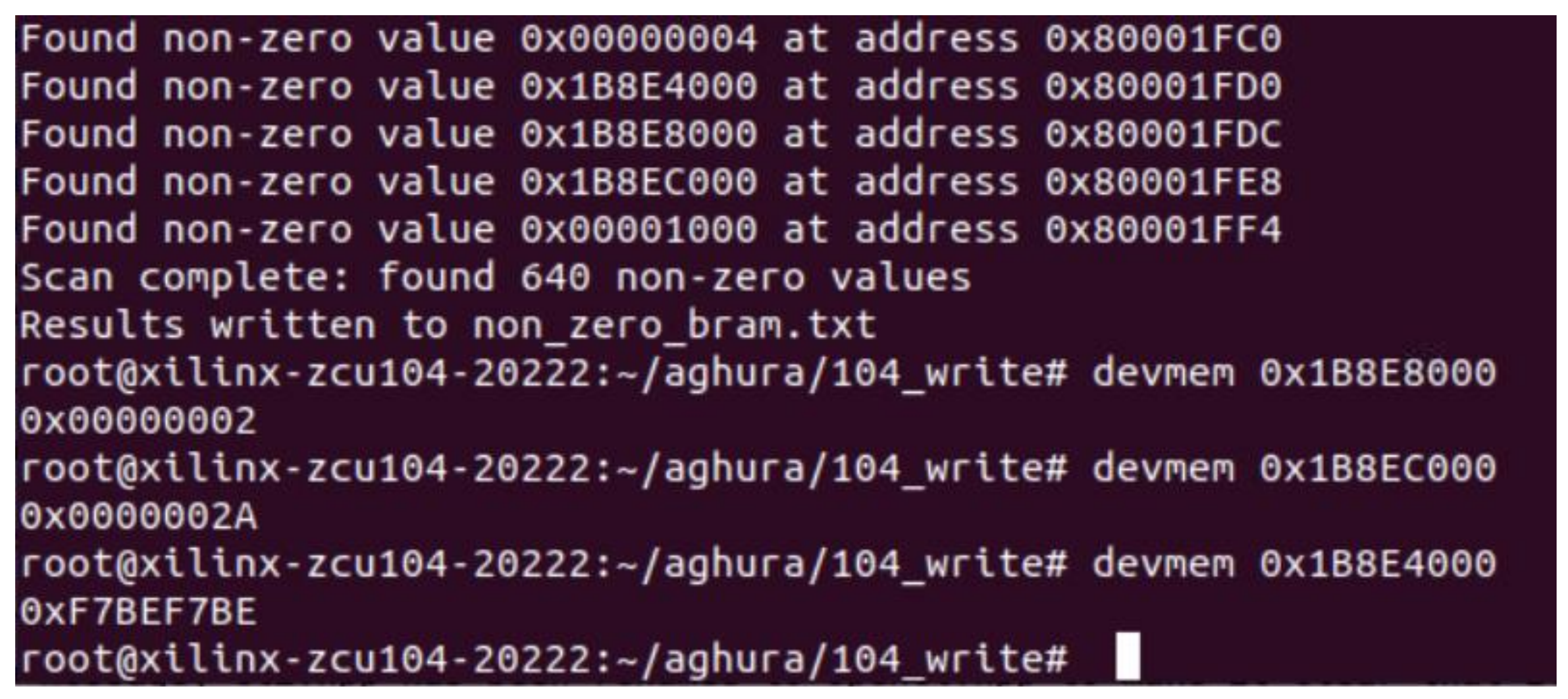
All experiments were conducted on a Xilinx ZCU104 board with root access, simulating a scenario where users have sequential access to the same FPGA resources in a shared environment.

Results

Our experiments conclusively demonstrated that BRAM contents persist after kernel termination, as seen in Figure 4. Specifically:

1. After running the victim's vector addition kernel, we successfully recovered 640 non-zero values, as opposed to the original 128 non-zero values after the memory zeroing phase (Figure 2).
2. The recovered values included 0x00000004, 0x00001000, 0x1B8E4000, 0x1B8E8000, and 0x18BEC000, which correspond to local variable values and global memory pointers referencing the matrix input/output values processed by the kernel.
3. Since DRAM data also persists after kernel execution [1], we were successfully able to read memory locations referenced by the BRAM pointers and retrieve sensitive user information inputted to and outputted by the kernel.
4. We observed that multiple runs produced consistent patterns of data persistence, indicating that this vulnerability is reliable and reproducible rather than probabilistic.

These findings confirm that sensitive data processed by one user remains fully accessible to subsequent users, creating a serious security breach in shared computing environments.



```
Found non-zero value 0x00000004 at address 0x80001FC0
Found non-zero value 0x1B8E4000 at address 0x80001FD0
Found non-zero value 0x1B8E8000 at address 0x80001FDC
Found non-zero value 0x1B8EC000 at address 0x80001FE8
Found non-zero value 0x00001000 at address 0x80001FF4
Scan complete: found 640 non-zero values
Results written to non_zero_bram.txt
root@xilinx-zcu104-20222:~/aghura/104_write# devmem 0x1B8E8000
0x00000002
root@xilinx-zcu104-20222:~/aghura/104_write# devmem 0x1B8EC000
0x0000002A
root@xilinx-zcu104-20222:~/aghura/104_write# devmem 0x1B8E4000
0xF7BEF7BE
root@xilinx-zcu104-20222:~/aghura/104_write# █
```

Figure 4: Reading BRAM after kernel execution. 512 new values found since memory zeroing. 0x80001FDC found to store pointer to vector input of second vector (0x02), 0x80001FE8 found to store pointer to vector addition output (0x2A = 42), 0x80001FF4 found to store value of local variable DATA_SIZE (0x1000 = 4096)

Conclusion

This research exposes a significant security vulnerability in FPGA computing platforms where BRAM contents persist between different users' kernel executions. The ability to extract another user's data represents a serious side-channel attack vector in multi-tenant environments. We recommend implementing automatic BRAM sanitization after kernel termination to prevent unauthorized access.

Acknowledgements: This research was made possible by the guidance of Prof. Sandip Kundu and prior lab work on BRAM persistence in Intel-based FPGAs by Chandra Sekhar Mummidi. A very special thanks to Bharadwaj Madabhushi for his continued support in troubleshooting issues, creating a custom PetaLinux image, and providing access to his personal machine and the ZCU104 board.

[1] B. Madabhushi, S. Kundu, and D. Holcomb, "Memory Scraping Attack on Xilinx FPGAs: Private Data Extraction from Terminated Processes," *arXiv preprint*, arXiv:2405.13927 [cs.CR], 2024. [Online]