Performance Analysis of the NEORV32 CPU Implementation on a SmartFusion2 FPGA with BRAM and Flash Memory

Tycho Schurink
University of Twente
S2556154

***Abstract*—Field-Programmable Gate Arrays (FPGAs) offer significant flexibility in embedded systems due to their reconfigurability, making them ideal for critical applications such as aerospace. However, the choice of memory technology used in FPGA-based systems can significantly influence both performance and radiation resilience.**

This study explores the impact of using two different memory types, Block RAM (BRAM) and Flash memory, on the performance of the NEORV32, an open-source RISC-V processor, implemented on a SmartFusion2 FPGA. The objective is to evaluate whether the performance advantages of BRAM outweigh its reduced radiation resistance compared to Flash memory in space-exposed environments.

To assess the impact of these memory types on performance, a series of benchmarks were conducted, including tests with different clock speeds, data sizes, and varying CPU setups. The results show that BRAM offers higher performance compared to Flash memory. This work provides insights into the trade-offs between performance and reliability in FPGA-based systems, offering valuable considerations for embedded system designers in radiation-prone environments.

***Index Terms*—Processor, Embedded Systems, System-on-Chip (SoC)**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) are highly versatile integrated circuits, offering reconfigurable logic suitable for various applications. This flexibility allows an FPGA to be reprogrammed and repurposed, for example, from performing voice recognition tasks one day to functioning as a processor in a satellite the next. This ease of reprogramming also makes it effortless to update the code if a bug is found inside the hardware, unlike Application-Specific Integrated Circuits (ASICs), where once a bug is found it cannot be fixed with a hardware reconfiguration.

For these reasons, FPGAs are often used in embedded systems, especially in fields like aerospace, where reconfigurability and rapid updates are critical. However, the choice of baseline technology used within FPGA systems can significantly impact both performance and resilience to radiation, which is crucial in environments exposed to cosmic rays and other radiation sources.

There are two main types of FPGAs, namely SRAM- and flash-based FPGAs. The former use static random-access memory (SRAM) cells to store the configuration bits of the programmable logic gates while the latter uses non-volatile flash memory cells.

FPGAs typically rely on two types of on-chip memory: either random-access memory cells (such as on-chip Block RAM, BRAM, or external RAM modules) or Flash cells to implement instruction and data memory. BRAM is a high-speed memory that provides fast access times, making it ideal for performance-critical applications. However, BRAM cells are volatile and tend to be more susceptible to radiation effects, such as Single Event Upsets (SEUs), which can alter data bits, especially in high-radiation environments like space.

In FPGA architectures, Flash-based FPGAs offer non-volatile configuration memory, providing enhanced resilience to radiation effects compared to SRAM-based FPGAs. However, this increased radiation tolerance often comes at the cost of slower access times. For instance, a study evaluating the radiation resilience of a Flash-based FPGA with a soft RISC-V core found that while Flash memory's non-volatility contributes to its robustness against radiation-induced errors, it generally exhibits slower access speeds compared to volatile memories like BRAM [1].

The NEORV32 CPU, an open-source soft RISC-V processor core, was chosen for this study due to its portability and the flexibility it provides. RISC-V is an open-standard instruction set architecture (ISA) that has gained popularity in embedded systems and academic research due to its modular design [2], [3], [4]. Which enables the implementation of various extensions for specific tasks. The NEORV32 core supports a base integer instruction set (RV32I) and can be configured with optional extensions, enabling a balance between performance and resource utilization in constrained design spaces. This modularity makes it ideal for testing configurations on an FPGA, as it can be tailored to meet specific performance requirements, such as floating-point operations or compressed instruction support.

Given the critical nature of memory type in determining both the performance and radiation resilience of FPGA-based systems, this study compares the performance of the NEORV32 CPU on an FPGA using BRAM as well as using an on-chip flash module as memory for the core. The goal is to understand whether the potential performance advantage of using BRAM justifies its reduced radiation resistance in applications that may face exposure to cosmic rays and other radiation sources.

II. Methodology

A. NEORV32 processor setup

In this project, the NEORV32 was configured with multiple RISC-V ISA extensions to test the impact of performance optimizations. The RISC-V ISA itself is structured to support custom extensions, making it an efficient choice for embedded applications. The standard NEORV32 core can be extended with various ISA subsets, including but not limited to:

- M-extension for integer multiplication and division, which improves execution speed for computation-heavy algorithms with a lot of multiplication operations.
- C-extension for compressed instructions, which reduces memory usage and can improve performance by decreasing the instruction fetch time.
- Zfinx-extension for floating-point operations using integer registers, which enables floating-point functionality on systems with limited memory.

These extensions are particularly useful when configured on an FPGA, as they allow for testing the CPU's adaptability and resource efficiency under different configurations, especially when comparing the memory types' impact on performance. The official repository for the processor was used and incorporated into this projects repository as a submodule and only one specific version identified by commit `cfff523c` was used [5].

B. Implementation on the SmartFusion2 FPGA in Libero

The SmartFusion2 (M2S010-VF400) FPGA was selected for this implementation. It is a flash-based FPGA from Microsemi. It has a maximum logical elements of 12084, which contains both 4-input lookup tables (4LUTs), and D-Flip flops (DFFs) [6]. Additionally, the fpga has a separate flash module. The software used to put the NEORV32 CPU onto the FPGA, is Libero SoC Design suite (Libero) from 2023 [7]. Using Libero enabled the creation and design of a working system, which it can then synthesize, generate the bitstream for and use to program the FPGA. Afterwards a serial port terminal (gtk-term) together with a UART device was used to debug and output data like the benchmark results. See the diagram in fig. 1.

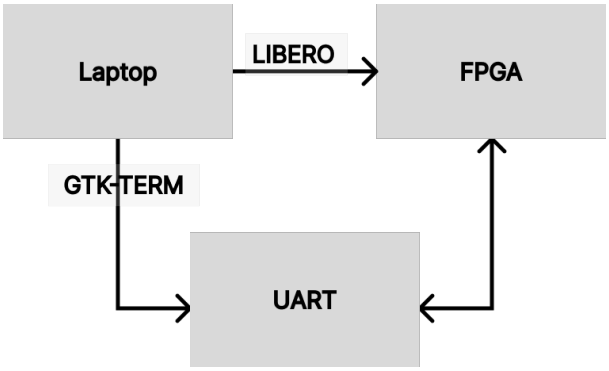
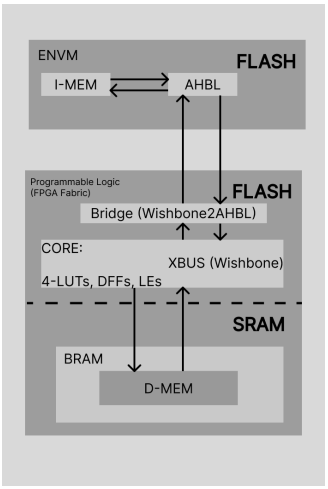
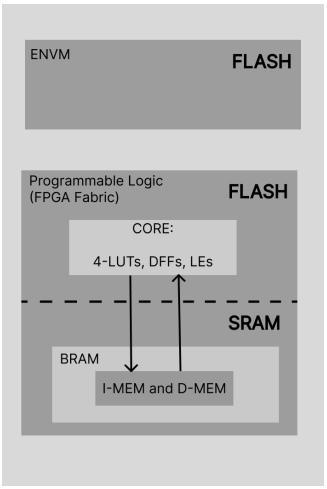


Fig. 1: Connection diagram

VHDL or VHSIC-HDL (Very High Speed Integrated Circuit Hardware Descriptor Language) is the programming language that is often used to create such systems. One can use this language to program entities, which can have many ports and generics to interact with other entities, additionally it can have an architecture in which the functionality of the entity as a whole is described. Libero's user interface is then used to connect the correct output to the input ports. However, there is also another way of doing this; using `.tcl` files. These files can be used to run Libero's actions from simple bash scripts, such as generating the bit-stream. Thus changes could be made without relying on Libero's GUI. This was better for the repository as there would have been a lot of extra files on there that could have been generated. Moreover this also made debugging more straightforward, as the bug was now quite easy to reproduce. Only 4 files needed to be checked, instead of 30+ files.
In figs. 2a and 2b high level diagrams of the implementation for flash and BRAM are shown respectively.



(a) Flash implementation diagram



(b) BRAM implementation diagram

C. Loading software into memory

All benchmarking was conducted using the previously mentioned FPGA, and the FPGA was reset and reprogrammed for each benchmark to ensure consistency across measurements. For the Flash implementation, the NEORV32 program (compiled benchmark) needed to be loaded into memory as a binary file containing 32-bit machine code instructions, one instruction per line. A python script that converts the hex file output by the RISC-V compiler into the required binary format was created to achieve this. This script facilitated the process of preparing the program for loading into the Flash memory. For the BRAM implementation, the procedure differed because BRAM is internal to the FPGA's Programmable Logic (PL) fabric. In this case, the program needed to be embedded as part of a VHDL module. To accomplish this, the VHDL file generated by the RISC-V toolchain, which represents the program as VHDL code, was copied directly into the project.

This step ensured the program was properly integrated into the FPGA design.

D. Benchmarking

To evaluate the effectiveness of using BRAM or Flash to implement the NEORV32 memory on the FPGA, a series of benchmarks was conducted with different configurations for the core. The NEORV32 CPU comes with two options for benchmarking, either Dhrystone or Coremark [8], [9]. At first Coremark was chosen since it is newer and an improvement on Dhrystone, addressing several of its limitations, such as the reliance on outdated workload patterns and susceptibility to compiler optimizations that distort performance results.
Furthermore, Coremark provides a standardized way to measure the processing power of embedded systems. Unlike other benchmarks that may focus on general-purpose processing tasks, Coremark is specifically designed for embedded environments. It tests integer math operations, control structures, and memory access patterns, which are all relevant to evaluating the NEORV32 CPU's performance on different memory configurations.
Due to the FPGA resource constraints, the Coremark code could not fit into the BRAM. Instead, a custom benchmark was created to focus on both instruction- and data-stressing. The "hardware performance monitor (HPM) demo" program provided in the NEORV32 repository served as a baseline, and the dummy benchmarking code was adjusted to meet specific requirements [5]. Instruction stressing involved generating a variety of operations such as addition, multiplication, bitwise operations, and bit-shifting. Data stressing was achieved by utilizing large arrays and performing various operations, including copying arrays, accessing arrays sequentially, and accessing them randomly. These operations were combined into a loop, with each loop cycle referred to as one iteration. Different numbers of iterations were tested initially, and it was observed that benchmarking scores did not change significantly when using fewer iterations. Therefore, 1000 iterations were used consistently for all tests. The final parameters include CPU clock speed, CPU setups, and data size, as detailed in table I.
The benchmark includes three `#define` macros: data size, instruction size, and copy size, this way the benchmark can be adjusted easily. Future tables will reference the data size, but all three are set to the same value.
The instruction size macro was used to create three arrays of that size. The previously mentioned operations, were then applied for each element in those arrays. The macro data size also used to create an array, then each element was initialized with the index from itself. Afterwards, it was used to randomly access some element from that array. Finally the copy size macro is used to create 2 arrays. Then one array is then copied to the other one, this is also done copy size amount of times. Finally the macro data size is

For this testing, two CPU setups were used (Standard and Performance) . With the performance setup, the performance-related RISC-V CPU extensions: `M`, `C` and `Zfinx` which provide specific hardware support for multiplication and division, compressed instructions, and floating-point operations using integer registers, respectively were enabled. Additionally, the generics `FAST_MUL_EN` and `FAST_SHIFT_EN` were both enabled to increase the speed of multiplication and shifting operations using DSP slices and fast barrel shifters for multiplication and shift operations respectively.
As for the standard setup, all previously mentioned extensions were disabled. Both setups do have the extensions `Zicntr` and `Zihpm` enabled and `HPM_NUM_CNTS` set to 13. This is done to enable the NEORV32 hardware performance monitors and counters to log the benchmark results.

TABLE I: Benchmarking parameters

Clockspeed [MHz]	CPU Setup	Data Size
10	Standard	100
50	Performance	200
100	-	400

III. Results

A. Flash

The results of the custom benchmark for the Flash implementation, summarized in table II, reveal key insights into the performance of the NEORV32 CPU with the benchmarking parameters (table I). The primary performance metric, iterations per second (Iterations/s), demonstrates a clear trend where higher values indicate improved throughput. Notably, the performance setup consistently outperforms standard, albeit with a relatively modest margin, on average 8.980%.
An examination of the relationship between data size and throughput indicates that doubling the data size leads to an approximate halving of iterations per second.
The influence of clock speed on performance is also evident, with higher clock speeds resulting in substantial increases in iterations per second. For instance, the throughput at $100\,\mathrm{MHz}$ is significantly greater than at $10\,\mathrm{MHz}$, reflecting the direct correlation between clock frequency and processing capacity. Importantly, while clock speed directly affects the overall performance, the relative difference between standard and performance setups remains consistent across frequencies, suggesting that the advantages of the performance setup are independent of clock speed.
Resource utilization, presented in section III-A, offers further insights into the implementation. Transitioning from standard to the performance setup incurs a noticeable increase in resource consumption across all categories. For example, at $100\,\mathrm{MHz}$, the utilization of 4LUTs increases from 38.65% in the standard setup to 59.96% in thee performance setup, while DFF utilization rises from 17.26% to 26.61%. These increases reflect the additional hardware components required to support the optimizations and extensions in the performance setup. However, the resource usage remains largely unaffected by clock speed.

TABLE II: Benchmark Results for Flash Memory

Clockspeed [MHz]	CPU Setup	Data Size	Iterations/s
10	Standard	100	15.410
		200	7.010
		400	3.110
	Performance	100	17.320
		200	8.110
		400	3.520
50	Standard	100	57.820
		200	27.710
		400	12.520
	Performance	100	63.210
		200	30.200
		400	13.620
100	Standard	100	82.420
		200	39.810
		400	18.700
	Performance	100	87.200
		200	41.610
		400	19.100

TABLE III: Resource Usage at 10, 50, and 100 MHz for Standard and Performance setups in %

Type	Setup	10 MHz	50 MHz	100 MHz
4LUT	Standard	37.44	37.49	38.65
	Performance	58.81	58.84	59.96
DFF	Standard	17.22	17.22	17.26
	Performance	26.56	26.56	26.61
I/O Register	Standard	0.00	0.00	0.00
	Performance	0.00	0.00	0.00
Logic Element	Standard	38.27	38.33	39.45
	Performance	60.26	60.08	61.25

B. BRAM

The benchmark results for BRAM memory are presented in table IV. The data indicates a consistent pattern where increasing the clock speed leads to a proportional increase, for the benchmark score, measured in iterations per second. At all clock frequencies, doubling the data size results in approximately halving the benchmark score, reflecting the anticipated scaling behavior due to memory access demands, just like in the Flash implementation.

Notably, there is no significant difference between thee standard and performance setups, as their results are identical across all configurations.

Resource utilization for BRAM is detailed in section III-B. As expected, the performance setup consistently requires higher resource allocation than the standard setup across all measured parameters. For instance, at 100 MHz, the performance setup uses approximately 88% of logic elements compared to 68% for the standard setup. Similarly, the utilization of 4LUTs is higher in the performance setup, with a difference of roughly 20% at maximum clock speed. These findings align with the trade-offs typically observed when optimizing for performance over resource efficiency.

It is evident that the performance setup for the BRAM implementation does not yield any performance gains over the standard setup. This is likely caused by the saturation of the CPU pipeline due to BRAM's high-speed memory access characteristics. The HPM data [10], reveals that with BRAM, the ALU experiences significant wait cycles (27.9% of total execution time) due to continuous instruction pressure from the memory subsystem. The single-cycle access time of BRAM creates a scenario where instructions arrive faster than the ALU can process them, effectively nullifying any potential benefits from the performance extensions. Conversely, the Flash implementation shows a 8.9% performance improvement when using the performance setup. This counter-intuitive result can be attributed to Flash memory's inherent wait states, which create natural gaps in instruction flow. These gaps allow the ALU to better utilize its performance extensions, as evidenced by the significantly lower ALU wait cycles (3.8% of total execution time) despite higher memory wait times (8.31% vs 4.10% for BRAM). The additional memory latency in Flash effectively acts as a natural pipeline balancing mechanism, preventing ALU saturation and allowing the performance extensions to improve overall processing efficiency.

Comparing both resource utilization, it is notable that the flash performance setup uses for almost all cases less resources than BRAM's standard setup implementation. Moreover, table VI is note worthy, as this table combines data from table II and table IV, to see how much more throughput BRAM can handle over Flash. It clearly shows that BRAM is more performant in all benchmarked scenarios. From a 20% increase in the worst case to 130% in the best case.

TABLE IV: Benchmark Results for BRAM Memory

Clockspeed [MHz]	CPU Setup	Data Size	Iterations/s
10	Standard	100	19.020
		200	9.010
		400	4.900
	Performance	100	19.020
		200	9.010
		400	4.900
50	Standard	100	97.430
		200	46.620
		400	21.910
	Performance	100	97.620
		200	46.910
		400	21.410
100	Standard	100	194.510
		200	93.000
		400	42.620
	Performance	100	194.510
		200	93.000
		400	42.620

TABLE V: Resource Usage at 10, 50, and 100 MHz for Standard and Performance setups in %

Type	CPU Setup	10 MHz	50 MHz	100 MHz
4LUT	Standard	64.95	65.29	67.34
	Performance	84.40	84.69	87.96
DFF	Standard	21.55	21.55	21.59
	Performance	29.73	29.73	29.77
I/O Register	Standard	0.00	0.00	0.00
	Performance	0.00	0.00	0.00
Logic Element	Standard	65.63	66.01	68.08
	Performance	85.29	85.64	88.83

TABLE VI: Average increase in throughput for BRAM compared to Flash implementation

Clockspeed [MHz]	Setups	Percentage Increase in Throughput
10	Standard	36.53
	Performance	20.04
50	Standard	70.58
	Performance	55.66
100	Standard	132.51
	Performance	123.24

IV. Conclusion

This study evaluated the performance and resource utilization of the NEORV32 CPU implemented on an FPGA using two different memory configurations: BRAM and Flash. Through a series of benchmarking tests, key insights into the impact of these memory types on performance and resource consumption were obtained.

The results showed that Flash-based implementations consistently delivered lower performance compared to BRAM, with throughput measured in iterations per second significantly higher when BRAM was used. This difference was particularly pronounced at higher clock speeds, where Flash memory's slower access times became more apparent. However, it is important to note that Flash's non-volatile nature provides a clear advantage in radiation resilience, making it more suitable for applications in radiation-prone environments, such as space-based systems [3].

In terms of resource utilization, the system with the instruction memory implemented in BRAM, particularly in the performance setup, exhibited higher consumption of logic elements and other resources compared to the system with the instruction memory implemented in an eNVM flash module. Interestingly, the Flash-based system using the performance setup utilized fewer resources than the BRAM-based implementation in the standard setup, highlighting the efficiency of Flash in terms of resource allocation despite its slower performance. Furthermore, resource utilization increases slightly with increased clockspeed, this is most likely due to higher clockspeed that require tighter timings. This in turn might lead to more complex routing, or duplication of logic to shorten critical paths. The benchmarking results also revealed that clock speed had a direct and proportional effect on performance, but the advantage of the performance setup over the standard setup remained relatively constant across both memory types, suggesting that the optimizations provided by the performance setup were not significantly affected by the memory configuration.

Overall, the choice between BRAM and Flash for FPGA-based systems is a trade-off between performance and radiation resilience. BRAM is ideal for high-performance applications where speed is critical, while Flash offers a robust solution for environments where resistance to radiation is paramount. Further studies could explore hybrid configurations that seek to balance these trade-offs and provide a more comprehensive solution for embedded systems requiring both high performance and radiation tolerance.

V. Recommendations

Given the findings of this study, several recommendations can be made for future work and for practitioners considering FPGA implementations with BRAM or Flash memory configurations.

While this study focused on performance analysis, future research should evaluate the radiation resilience of BRAM and Flash implementations under realistic conditions. This would provide critical data for space and aerospace applications.

Although Coremark benchmarking was successful for certain configurations and included in the appendix, its incompatibility with the BRAM implementation highlights areas for further investigation. Future work should aim to identify and resolve the underlying issues preventing full compatibility. Doing so could ensure a more comprehensive evaluation of performance across all memory configurations.

Moreover, other benchmarking suites, such as CPU SPEC or Embench [11] [12], can be used to better compare this implementation with similar research projects or FPGA-based designs.

By addressing these areas, future research and development could further optimize FPGA-based systems for a wide range of applications, particularly in space missions and other environments where both performance and resilience to radiation are crucial.

References

[1] K. Böhmer, "Radiation resilience evaluation of a flash-based fpga with a soft risc-v core," September 2023.

[2] R.-V. International, "Risc-v." urlhttps://riscv.org/. [Online] Accessed: 13-11-2024.

[3] K. Bohmer, B. Forlin, C. Cazzaniga, P. Rech, G. Furano, N. Alachiotis, and M. Ottavi, "Neutron radiation tests of the neorv32 risc-v soc on flash-based fpgas," in *Neutron Radiation Tests of the NEORV32 RISC-V SoC on Flash-Based FPGAs*, 2023.

[4] C. W. Peng, "Performance evaluation of risc-v microcontroller system on fpga: A study of the neorv32 core," 2024.

[5] stnolting et al, "Neorv32 processor." https://github.com/stnolting/neorv32, 2024. [Online] Accessed: 13-11-2024.

[6] Microsemi, "Smartfusion 2 fpgas." https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/smartfusion-2-fpgas. [Online] Accessed: 13-11-2024.

[7] Microsemi, "Libero." https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/fpga/libero-software-later-versions. [Online] Accessed: 13-11-2024.

[8] A. R. Weiss, "Dhrystone benchmark." https://www.eembc.org/techlit/datasheets/dhrystone_wp.pdf, October 2002.

[9] EEMBC, "Coremark." https://www.eembc.org/coremark/. [Online] Accessed: 13-11-2024.

[10] T. Schurink, "Neorv32 sf2 implementations." https://github.com/Tshadow999/neorv32_sf2_implementations, 2024. [Online] Accessed: 13-11-2024.

[11] J. B. et al., "Embench." https://www.embench.org/. [Online] Accessed: 13-11-2024.

[12] SPEC, "Spec cpu 2017." https://www.spec.org/cpu2017/. [Online] Accessed: 13-11-2024.

VI. Appendix

A. Code Repository

All code, configuration files, and benchmark scripts used in this thesis can be accessed at the following GitHub repository [10].

B. Custom benchmark code

```
for (int i = 0; i < ITERS; i++)   {
  // Instruction stressing
  volatile int a[INSTR_SIZE], b[INSTR_SIZE], c[
    INSTR_SIZE];

  for (int i = 0; i < INSTR_SIZE; i++) {
    b[i] = i;
    c[i] = i + 1;
  }

  for (int i = 0; i < INSTR_SIZE; i += 4) {
    int tmp1 = b[i] + c[i];
    int tmp2 = b[i + 1] ^ c[i + 1];
    a[i] = tmp1 * tmp2;
    a[i + 1] = tmp2 >> 2;
    a[i + 2] = (tmp1 | c[i + 2]) & 0xFF;
    a[i + 3] = b[i + 3] + (c[i + 3] << 1);
  }

  // Data stressing
  volatile int data[DATA_SIZE];
  for (int i = 0; i < DATA_SIZE; i++) {
    data[i] = i;
  }

  // Random access
  int sum = 0;
  srand(12345);
  for (int i = 0; i < DATA_SIZE; i++) {
    sum += data[(rand() ^ i) % DATA_SIZE];
  }

  // Memcpy stress
  volatile char src[COPY_SIZE], dest[COPY_SIZE];
  src[0] = a[DATA_SIZE - 3];
  for (int i = 0; i < COPY_SIZE; i++) {
    memcpy((void*)dest, (void*)src, COPY_SIZE);
  }
}
```

VII. Coremark Methodology

While other benchmark suites, such as SPEC CPU or Embench also provide comprehensive evaluations of embedded processors, adapting them for use with NEORV32 would have required substantial effort [12], [11]. These benchmarks are not natively compatible with the NEORV32 software library, and refactoring their code to align with NEORV32's specific architecture and toolchain would have been time-intensive. Given the practical constraints of this study, Coremark was a natural choice as it is readily supported by the NEORV32 ecosystem and aligns well with the goals of this performance analysis.

VIII. Coremark Results

A. Flash

The results, summarized in table VII, highlight the number of iterations per second achieved in both the Standard and Performance setups across clock speeds of 10, 50, and 100 MHz. Additionally, the Coremark scores provide a standardized measure of the CPU's performance under each configuration. The score is calculated by $\frac{\text{iterations}/s}{\text{MHz}}$, the higher the better [9]. As shown in the table, increasing the clock speed tends to improve the iterations per second, although the Coremark score shows a less consistent trend. The difference between Standard and Performance setups is also notable, particularly at lower clock speeds, where the Performance setup yields slightly higher iterations per second.

TABLE VII: Results for Flash memory with 1000 data size and 2000 iterations

Clockspeed [MHz]	CPU Setup	Iterations/s	Coremark Score
10	Standard	7.60	0.76
	Performance	8.27	0.83
50	Standard	22.78	0.46
	Performance	23.49	0.47
100	Standard	39.93	0.40
	Performance	40.80	0.41

TABLE VIII: Resource Utilization for Flash implementation

	Standard Setup		Performance Setup	
	Used	Percentage	Used	Percentage
4LUT	4126	34.14%	4819	39.88%
DFF	2176	18.01%	2492	20.62%
I/O Register	0	0.00 %	0	0.00 %
Logic Element	4266	35.30%	4974	41.16%

Table VIII shows the resource utilization for both Standard and Performance setups of the NEORV32 CPU on the SmartFusion2 FPGA. The Performance seetup consistently uses more resources than the Standard setup across all categories, due to the additional hardware optimizations enabled, such as the multiplication and shift extensions. The testing confirmed that varying the FPGA's clockspeed had no significant effect on resource utilization, as the logic elements remain constant regardless of operating frequency. This suggests that, while different setups impact hardware resource demands, changes in clockspeed alone do not affect the FPGA's overall resource allocation.