Journal of
Low Power Electronics and Applications

Article

Performance Evaluation of FPGA, GPU, and CPU in FIR Filter Implementation for Semiconductor-Based Systems

Muhammet Arucu [1,*] and Teodor Iliev [2]

[1] Department of Computer Technologies, Gönen Vocational School, Bandırma Onyedi Eylül University, Bandırma 10200, Türkiye
[2] Department of Telecommunications, University of Ruse, 7017 Ruse, Bulgaria; tiliev@uni-ruse.bg
* Correspondence: marucu@bandirma.edu.tr

Abstract

This study presents a comprehensive performance evaluation of field-programmable gate array (FPGA), graphics processing unit (GPU), and central processing unit (CPU) platforms for implementing finite impulse response (FIR) filters in semiconductor-based digital signal processing (DSP) systems. Utilizing a standardized FIR filter designed with the Kaiser window method, we compare computational efficiency, latency, and energy consumption across the ZYNQ XC7Z020 FPGA, Tesla K80 GPU, and Arm-based CPU, achieving processing times of 0.004 s, 0.008 s, and 0.107 s, respectively, with FPGA power consumption of 1.431 W and comparable energy profiles for GPU and CPU. The FPGA is 27 times faster than the CPU and 2 times faster than the GPU, demonstrating its suitability for low-latency DSP tasks. A detailed analysis of resource utilization and scalability underscores the FPGA's reconfigurability for optimized DSP implementations. This work provides novel insights into platform-specific optimizations, addressing the demand for energy-efficient solutions in edge computing and IoT applications, with implications for advancing sustainable DSP architectures.

Keywords: circuits; microelectronic; semiconductors; FPGA, GPU and CPU

check for updates

Academic Editors: Shoue Chen, Xiaolong Wang and Ke Xie

Received: 30 May 2025
Revised: 10 July 2025
Accepted: 16 July 2025
Published: 21 July 2025

Citation: Arucu, M.; Iliev, T. Performance Evaluation of FPGA, GPU, and CPU in FIR Filter Implementation for Semiconductor-Based Systems. *J. Low Power Electron. Appl.* **2025**, *15*, 40. https://doi.org/10.3390/jlpea15030040

1. Introduction

Semiconductors are materials, such as silicon and germanium (elemental semiconductors), or compounds like gallium arsenide, that are used to manufacture devices such as integrated circuits and microchips. After the invention of the transistor in 1948 and the definition of integrated circuit (IC) in 1958, the need for electronic circuit elements has increased the importance of semiconductor device technology [1,2]. Semiconductors represent the fundamental components of different integrated circuits, such as transistors and microprocessors. Indeed, they are the main components in the construction of logic gates and other digital circuits. In recent years, advances in semiconductor technology have enabled the development of smaller, faster, and more reliable electronic devices [3–5]. With the continued miniaturization of semiconductor devices, it has become possible to manufacture semiconductors with higher precision and quality, such as improved speed, functionality, and integration density, at reduced costs [6].

The resistance of the semiconductors is higher than that of conductors but lower than that of insulators; it is thus among the classes of conductive and insulating materials. When constructing integrated circuits, semiconductors composed of crystalline solids are suitable for use. Semiconductors, which are the basis for the electronics industry, have two types:

elements and compounds. Examples of compounds are zinc oxide (ZnO) and copper oxide (CuO). Silicon (Si), germanium (Ge), and selenium (Se) can be given as examples of elements. Electronic circuit components, computers, mobile phones, and digital audio players are made of semiconductor materials. Germanium and silicon are used in the production of diodes, transistors, and integrated circuits, which are the main elements of electronics [7]. Semiconductors are found in nature as simple elements as well as compound elements in the laboratory. Rapid advancements in semiconductor technology have paved the way for innovative solutions in various industries, including telecommunications, automotive, and artificial intelligence [8–10].

With the advances in semiconductor technology, CPUs, GPUs, and FPGAs have become smaller, faster, and more efficient. While CPUs have become more powerful in general-purpose use with multi-core structures, GPUs stand out in artificial intelligence and parallel computing. FPGAs, on the other hand, provide low latency and energy efficiency by offering application-specific hardware speed customization. While these three hardware types offer converging performances thanks to technological advances, each still excels in certain applications. This study compares the performance of FPGA, GPU, and CPU platforms in implementing a finite impulse response (FIR) filter, a fundamental task in digital signal processing (DSP). While FPGAs are often used for hardware verification and real-time processing, GPUs excel in parallel computing, and CPUs are designed for general-purpose tasks, the FIR filter serves as a standardized benchmark to evaluate computational efficiency, latency, and energy consumption across these heterogeneous platforms. By focusing on a common DSP application, this study highlights the architectural strengths and trade-offs of each platform, providing insights into their suitability for real-time and high-throughput applications in semiconductor-based systems. The paper is organized as follows: Section 2 reviews semiconductor technologies; Section 3 details the FIR filter methodology; Section 4 presents performance results; Section 5 discusses trade-offs, highlighting FPGA's energy efficiency, GPU's parallelism, and CPU's programming ease; and Section 6 concludes with contributions and future research directions.

2. Background and Related Work

2.1. Hardware Platforms

Semiconductors process stored, processed, and transported data much faster and use lower power consumption per bit. Key products derived from semiconductor manufacturing, which are essential for various electronic equipment and devices, include recordable disks, flash drives, hard disk drives, solid-state drives, and cloud-based storage solutions. Some industry components in the semiconductor market, such as products (ICs, discrete semiconductors, opto-electronics, and sensors), applications (communications, consumer electronics, automotive electronics, and computers), and artificial intelligence, will increase trends in the semiconductor industry in the near future. Typical examples are AI, IoT, autonomous vehicles, and chip security. Developing industrial technologies such as artificial intelligence, automotive electronics, IoT, and augmented/virtual reality (AR-VR) stand out in most production and manufacturing industries. At the heart of all these technologies are the chips that drive them. Industries more than ever rely on the semiconductor industry to ensure that all these technologies fulfill their duties and provide the computing power required.

In today's market dominated by technology, the semiconductor industry, which is the backbone of electronic devices, tops the list. Semiconductors are the source of creativity in health, military, computing, communication, transportation, and numerous other applications. As the expectations grow, the consumers' experiences are gradually improving in these technologies, where the productivity and consolidation increase. Semiconductor manufacturing is becoming increasingly competitive as consumers demand higher performance devices with more functionality [11,12]. Therefore, semiconductor manufacturers adopt digital manufacturing techniques to increase competitiveness and achieve higher operational excellence, which is depicted in Figure 1.

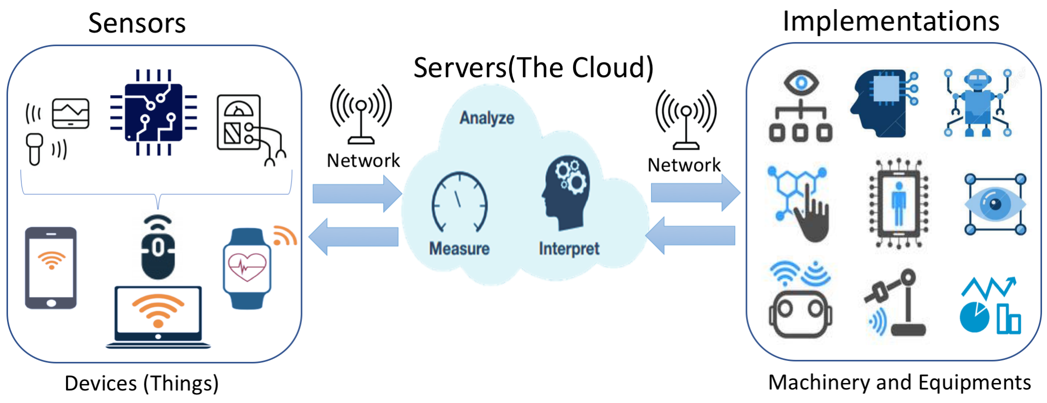


Figure 1. The Internet of Things (IoT) encompasses four main elements, each of which plays a critical role in the functionality and success of IoT systems. Key elements of IoT include four main elements: devices, sensors, networks, and implementations.

As high-performance CPUs with more than half a century of history are designed to efficiently process data in serial programming, they are also optimized for parallel programming through various existing techniques. GPU processors are largely parallel, made of smaller and more specialized cores than high-performance CPUs. GPU architecture is optimized for batch volume on all cores. It is especially used in areas where large datasets need to be processed within parallelizable processes. Due to the architecture of graphics processing units and the great number of cores, there is a large and widening gap between CPU and GPU performances.

FPGAs are programmable semiconductor circuits composed of customizable logic blocks in a matrix structure connected to each other with programmable connectors. FPGAs can be easily programmed for any application, and the previous FPGA contents may be changed and reprogrammed for different applications. FPGAs have quickly found a place in the industry thanks to their flexible architectural structures and programming features. Today, they are preferred in diverse application areas from automotive, telecommunication, and space applications to the defense industry, and particularly in many application areas that require flexibility with high performance. Unlike CPUs and GPUs, which are software-programmable fixed architectures, FPGAs are reconfigurable with user-defined computational engines. When writing a software tool targeting an FPGA, the compiled instructions become hardware components organized on the FPGA structure, and all of these components can run in parallel.

2.2. Overview of CPU, GPU, and FPGA Architectures

As communication becomes faster and more secure, integrated semiconductor devices provide stable and efficient communication. In the development of wireless communication systems like 5G, the use of System-on-Chip (SoC) and FPGA devices is becoming increasingly prevalent. Unlike a processor that executes a program, the FPGA configures itself to be a work circuit that will then respond to inputs in a way that a special piece of hardware will behave. Unlike CPUs, FPGAs do not have a fixed hardware design. In contrast, they can be programmed from scratch according to user applications. The functions to be performed by the logical cells and the connections between these functions are determined by the user. For this reason, the arithmetic operations that the programmer uses must be coded from scratch while programming the FPGA [13,14].

Additionally, mobile Wi-Fi chips, cellular baseband processors, Bluetooth transceivers, near-field communication (NFC) chips, global positioning system (GPS) receivers, and similar components can be categorized as semiconductors used in wireless communication [15,16]. Thus, it reveals the importance of semiconductors in the wireless communication industry. It is possible to obtain a high level of convenience, comfort, and energy efficiency with IoT products. These devices rely on compact microchips designed for high-speed data exchange, with each chip uniquely engineered to fulfill its specific function. In the context of automotive vehicles, sensors play a crucial role during operation, and even insurance companies utilize sensor-generated data to assess safe driving behaviors. Integrated circuits are primarily designed for drones used in control, distribution, surveillance, and mapping. Semiconductor technology is used in related sensing devices, for example, medical devices used in hospitals, electrocardiogram (ECG or EKG) machines, and communication devices [17–19]. In addition, the semiconductor industry has gained significant place from the growth of IoT in the construction of high-end devices such as cell phones, laptops, electric vehicles, and wearables [20,21].

The usual approach for modeling or calculating a task is to write some code in a CPU or GPU-based architecture. In FPGAs, a special circuit is designed for this particular calculation. There are various publications in the literature with a focus on comparisons of FPGA, CPU, and GPU applications, and the results may vary with respect to the platforms implemented [22]. FPGAs are integrated circuits with a programmable hardware structure. Unlike GPUs or ASICs, the circuitry inside an FPGA chip can be reprogrammed when required. This capability makes ASICs an excellent alternative, which requires a long development time and significant investment to design and manufacture FPGAs. FPGAs, CPUs, and GPUs are three distinct processing technologies, each with unique strengths and applications [23].

Numerous studies have explored the comparative performance of FPGAs, CPUs, and GPUs across various applications. In summary, GPUs excel in highly parallel workloads, outperforming CPUs, while FPGAs demonstrate superior energy efficiency, particularly in tasks like deep learning inference [24]. FPGAs consume less power than GPUs and CPUs, making them ideal for real-time and edge computing applications [25]. They are best suited for low-latency tasks, GPUs for compute-intensive parallel workloads, and CPUs for general-purpose sequential processing [26]. Additionally, GPUs offer scalability for large-scale parallelism, whereas FPGAs provide hardware-level customization for tailored application-specific solutions [27]. This comparative understanding provides information on how to use the strengths of each technology to address real-world challenges in IoT, AI, autonomous systems, and edge computing. Future research should focus on hybrid approaches that combine these technologies to achieve optimal performance and efficiency [28].

2.3. FIR Filter Fundamentals

FPGAs, CPUs, and GPUs implement FIR filters in distinct ways, with FPGAs offering low-latency, highly parallel hardware solutions for real-time applications, CPUs providing flexible and sequential processing for general-purpose tasks, and GPUs leveraging massive parallelism to accelerate computationally intensive FIR filtering operations in high-throughput scenarios. Filters have an important role in digital signal processing. Digital filters are used in most electronic devices. Therefore, digital filters can be implemented in both hardware and simulation. In recent years, it has been shown that the use of FPGAs has increased in the implementation of real-time and high-performance signal processing applications [29]. FPGA chips, which were initially developed only for controlling digital designs, have gained the ability to operate in parallel with new technological developments. It is known that the power consumption of designs with an FPGA is low, with a relatively high working speed. In addition, arithmetic operations are easily performed with an FPGA. In order to clearly clarify this information, an FIR filter design was performed in the Vivado HLX environment, and the design was implemented in the Xilinx ZYNQ XC7Z020-1CLG400C FPGA environment. The designed filter has been implemented using the Verilog hardware description language, leveraging the gate arrays and other hardware resources available on a FPGA. The FPGA's resources, including configurable logic blocks, dedicated DSP slices, and on-chip memory elements, are utilized to efficiently realize the filter's functionality through precise configuration and interconnection, enabling high-performance digital signal processing within the hardware constraints of the target platform. The processing time of the FIR filter is analyzed across CPU, GPU, and FPGA implementations to evaluate their performance and suitability for various application scenarios.

In evaluating the performance of FIR filter implementations, several key parameters such as latency, resource utilization, and power efficiency must be considered. While CPUs offer ease of programmability and are suitable for a wide range of applications, they often fall short in meeting the stringent timing requirements of real-time systems due to their inherently sequential architecture. GPUs, on the other hand, exploit data-level parallelism through thousands of cores, making them well-suited for batch processing and high-throughput applications; however, their high power consumption and latency in memory access can be limiting factors. FPGAs strike a balance by offering a customizable hardware fabric that enables fine-grained parallelism, pipelining, and low-latency computation tailored to specific application needs. These characteristics make FPGAs particularly advantageous for embedded systems and real-time digital signal processing tasks. Consequently, the comparative analysis of these platforms provides critical insights into selecting the most appropriate hardware architecture for FIR filter deployment based on specific performance requirements and system constraints.

3. Methodology

In order to accelerate the functions commonly used on large datasets in digital signal processing, calculations are performed using multicore processors, graphics processors, FPGA-based systems, and ASIC designs. Many different types of accelerator methods are used, including CPUs, GPUs, FPGAs, and ASICs, to improve computing performance and energy efficiency in application areas [30]. While ASICs are used only to fulfill a specific process and task, FPGAs are used mostly in parallel work applications. FPGAs show significant advantages in real-time data processing over GPUs and other types of hardware in certain use cases. FPGAs ensure a low level of latency against GPUs and CPUs. FPGAs and ASICs are faster than GPUs and CPUs as they work on bare metal and have no operating system [14,31]. In the applications designed in our study, the performances of different hardware platforms were compared with various variables. To see the advantages of FPGAs, it is necessary to compare them with alternatives of processing units such as CPUs and GPUs in terms of flexibility and efficiency. GPUs are ideal for rendering screenshots, animations, videos, and handling graphics-intensive loads. Found in computers, phones, and other graphics processing devices, GPUs are often used today to relieve a CPU's workload. FPGAs have a matrix of logic blocks configurable via programmable interconnects for specific applications or functions. In order to form the basis of our research, GPU, FPGA, and CPU platforms were examined in terms of performance, power efficiency, and usability.

Finite impulse response filters are designed using various methods, each with distinct computational requirements and suitability for hardware platforms. The Kaiser window method, used in this study, balances main lobe width and side lobe attenuation through a shape parameter (β), making it suitable for FPGA implementations due to its flexibility in coefficient computation [24]. Other methods include the Hamming and Blackman windowing techniques, which are computationally simpler and often preferred for CPU-based implementations due to their straightforward calculations. The Parks–McClellan algorithm, which optimizes filter coefficients for minimal error, is well-suited for FPGA and GPU platforms where parallel processing can accelerate complex computations [29]. Frequency sampling methods, while less common, are effective for GPUs due to their reliance on fast Fourier transform (FFT)-based operations. This study adopts the Kaiser window method for its balance of performance and design flexibility across all platforms, enabling a fair comparison of FPGA, GPU, and CPU implementations.

While this study uses FIR filter implementation as a benchmark to evaluate FPGA, GPU, and CPU platforms, the scientific literature reports several other methods for assessing digital processing devices. FFT is commonly used to evaluate frequency-domain performance, particularly on GPUs due to their parallel processing capabilities [17]. Infinite impulse response (IIR) filters, which involve recursive computations, are another benchmark suitable for CPUs and FPGAs [18]. For AI-related applications, matrix multiplication and convolutional neural networks (CNNs) are frequently used to compare computational throughput and energy efficiency across platforms [30]. These alternative methods complement the FIR filter benchmark by highlighting different aspects of platform performance, such as memory bandwidth for FFT or recursive computation efficiency for IIR filters.

Filters are generally used in electronic circuits for various purposes such as filtering noise and unwanted signals, separating certain frequencies, and limiting signals before sampling. A digital filter is a method or algorithm that works on digitized analog signals, converting the input signal to the desired output signal. The main design purposes of the filters, which have very wide application areas, are to separate the interfered signals from each other, to increase the signal quality by reducing the noise in the signal, and to recover the distorted signal. Digital signal processing involves performing the desired operations on the frequency spectrum of a signal to provide the desired result after these operations. This digital signal processing structure is called a digital filter. Digital filters can be implemented with either software or hardware. Digital filters are divided into two categories according to their impulse responses. These are finite pulse response and IIR filters [32,33]. The Parks–McClellan algorithm is an iterative approach to explore the ideal Chebyshev FIR filter. It is used to design and apply effective FIR filters. An indirect method is used to find the optimal filter coefficients [34]. The input and output relation of the FIR filter given above can be given as in Equation (1);

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots\ldots + b_N x[n-N] = \sum_{i=0}^{N} b_i . x[n-i] \tag{1}$$

where $x[n]$ represents the input signal and $y[n]$ represents the filter output.

Digital signal processing techniques are widely used in many applications, such as communication and multimedia. Filters have an important use in digital signal processing. Digital filters are used in most electronic devices. Therefore, digital filters can be implemented not only with simulations but also with hardware. In recent years, it has been observed that the use of FPGAs in real-time signal processing applications that require high performance has increased. FPGA chips, initially developed for controlling only digital designs, have the ability to work in parallel with the new developments in technology. It is known that FPGA-realized designs have low power consumption and high operating speed, as seen in Figure 2. In order to reflect the developments in semiconductor technology, the above mathematical model was implemented on the CPU, GPU, and FPGA, and comparisons were made in terms of time and acceleration capacities.

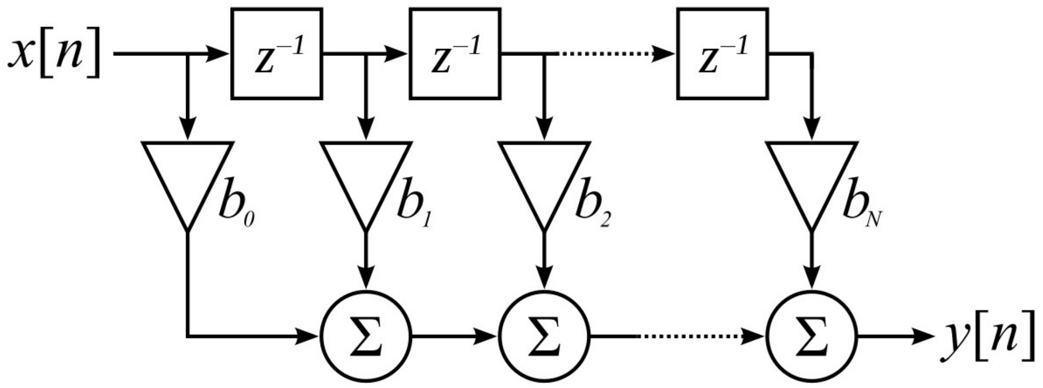


Figure 2. Block diagram of digital FIR filter. A direct form discrete-time FIR filter of order N. The flow diagram of an FIR filter is shown below; $x[n]$ and $y[n]$ are inputs and outputs, respectively. z^{-1} is the unit delay, and b_N are the filter coefficients.

4. Performance Evaluation and Results

To design an FIR filter, the Kaiser window method is a widely adopted technique due to its flexibility in controlling the trade-off between main-lobe width and side-lobe attenuation. This method allows for the precise specification of filter characteristics by adjusting the shape parameter (β), making it suitable for applications requiring stringent frequency response criteria. The Kaiser method is widely used to calculate the filter coefficients that give a specified frequency response. Using this method, the steps and calculations required to design a filter with the specified properties are detailed below. Frequency Ranges and Gains:

- Passband: 0 Hz–5 MHz;
- Stopband: 10 MHz–50 MHz;
- Passband: 5 MHz–10 MHz (this is a range that does not exist for an ideal filter, but in practice, it is taken into account in the design).

Kaiser Window Parameters: Kaiser window is used to calculate the filter length (N) and the beta (β) parameter. Beta is chosen depending on the desired stopband attenuation in the filter design. The value of beta is determined by the value of attenuation in decibels.

Calculation Stages: Filter Length (N):

$$N \geq \frac{A - 8}{23.703 \cdot \Delta f} \tag{2}$$

Here, A is the attenuation value in dB. Δf is the frequency range normalized by f_s (sampling frequency).

In this case, $A = 40$ dB and $\Delta f = \frac{f_{stop} - f_{pass}}{f_s} = \frac{10\,\text{MHz} - 5\,\text{MHz}}{100\,\text{MHz}} = 0.05$

$$N \geq \frac{40 - 8}{23.703 \cdot 0.05} \approx 27$$

It is designed with 27 taps, so it can be taken as $N = 27$.

Beta (β) Parameter:

$$\beta = \begin{cases} 0.1102 \cdot (A - 8.7), & \text{if } A > 50, \\ 0.5842 \cdot (A - 21)^{0.4} + 0.07886 \cdot (A - 21), & \text{if } 21 \leq A \leq 50, \\ 0, & \text{if } A < 21. \end{cases} \tag{3}$$

In this case, since $A = 40$ dB,

$$\beta = 0.5842 \cdot (40 - 21)^{0.4} + 0.07886 \cdot (40 - 21) \approx 5.65$$

Calculating Filter Coefficients:
The filter coefficients $h(n)$ with the Kaiser window are calculated as follows:

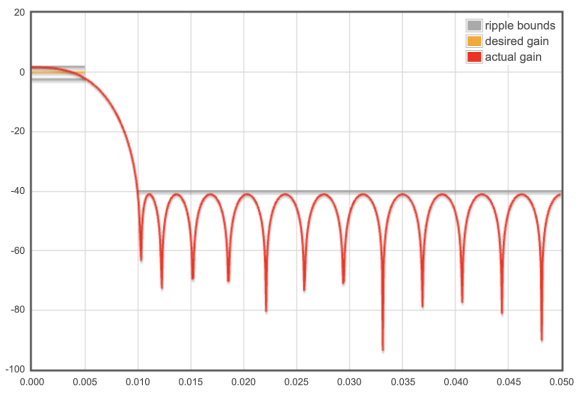
$$h(n) = \frac{I_0\left(\beta\sqrt{1 - \left(\frac{2n}{N}\right)^2}\right)}{I_0(\beta)} \cdot h_{ideal}(n) \tag{4}$$

Here, I_0 is the modified Bessel function, and the ideal filter coefficients are calculated by the sinc function for the ideal low-pass filter:

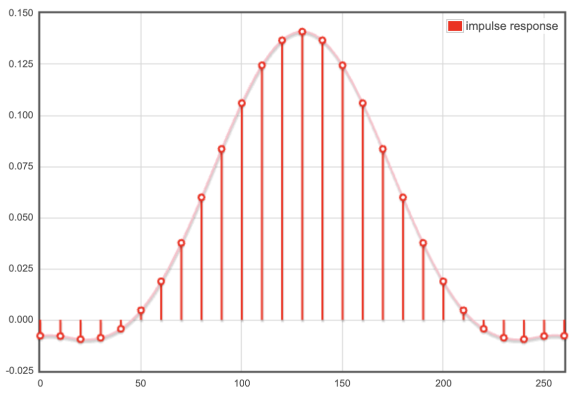
$$h_{ideal}(n) = \frac{\sin\left(2\pi f_c\left(n - \frac{N-1}{2}\right)\right)}{n - \frac{N-1}{2}} \tag{5}$$

Here, f_c is the cut-off frequency, and 5 MHz is used as f_s .

Filter coefficients are calculated using the filter design application available in [35] for 0 mHz–5 mHz bandpass and 10 mHz–50 mHz band stop frequencies, which is shown in Figure 3. The frequency response of the analyzed system provides crucial insight into how the system reacts to different input frequencies, showcasing its stability and performance characteristics across a broad spectrum. Meanwhile, the impulse response highlights the system's reaction to a sudden input, allowing for the assessment of its dynamic behavior and time-domain properties, which are vital for accurate system modeling and control.



(**a**) Gain Frequency



(**b**) Impulse Response

Figure 3. The figures illustrate the gain frequency characteristics and impulse response of the designed FIR filter.

FIR filters are mathematically fast and relatively easier to design and implement digitally. The determined coefficients are added to the FIR filter software. A hardware filter is designed on the FPGA, with a software filter running on the CPU and GPU. The FIR compiler module was added to the block diagram on the Xilinx Vivado HLS program, and the t-filter coefficients and FIR filter settings were added to the module. The advanced extensible interface (AXI) communication protocol, one of the most important features of Xilinx FPGAs, provides data communication between the ARM CPU and FPGA on the PYNQ Z1. AXI direct memory access module has been added to the FIR compiler module as shown in Figure 4. This module provides a detailed analysis of the integration and functionality of the FIR compiler and the AXI direct memory access module within an FPGA-based system. The FIR compiler is employed for efficient digital filtering, while the AXI direct memory access module facilitates high-speed data transfer between memory and processing units. Their combined use enhances the performance of real-time applications by leveraging hardware acceleration and optimized data handling.

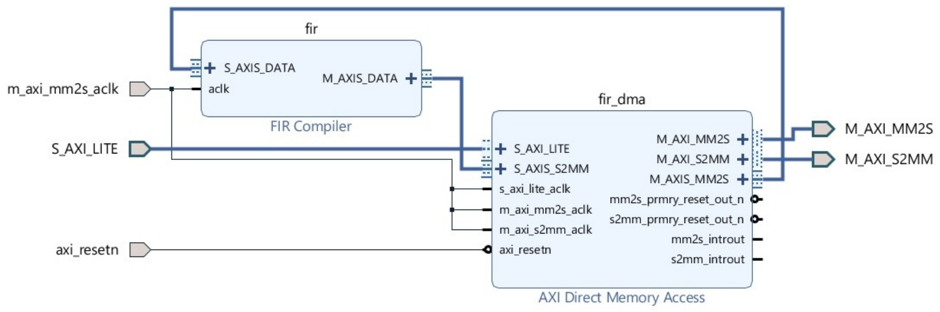


Figure 4. FIR compiler and AXI direct memory access module are critical components in the design of digital signal processing systems.

The FIR compiler and AXI DMA modules are combined into a single, unified module identified by a specific filter name, streamlining signal processing tasks and enhancing system efficiency by integrating filter design and high-speed data transfer capabilities. The designed filter module is combined with the ZYNQ processing system module and AXI modules in order to communicate with the processor on the PYNQ board and to work with Python codes, as shown in Figure 5. Thus, the output signal results were examined by sending the input signal to the hardware filter designed on the FPGA using the Linux operating system and Jupyter Notebook (https://jupyter.org/) that we installed on PYNQ board. The following figure shows the block diagram of the FIR filter designed on the FPGA with the VIVADO HLX program, the ZYNQ operating system, and the hardware that performs bitstream communication between these two systems.

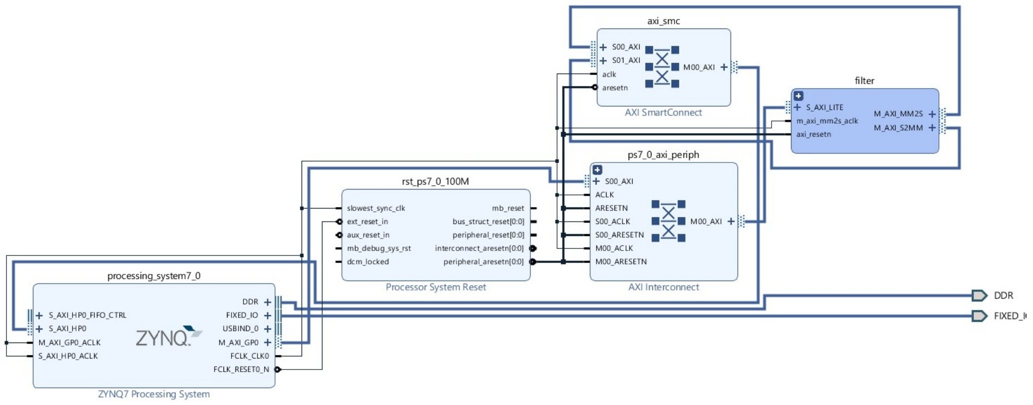
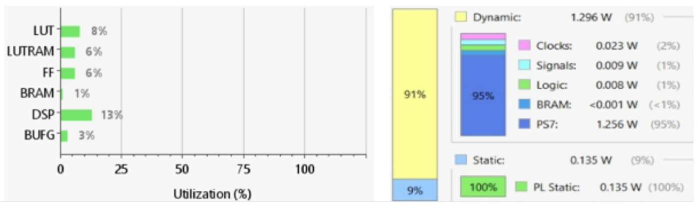


Figure 5. FPGA hardware filter and ZYNQ operating system block diagram. Here is the block diagram of an FPGA hardware filter integrated with a ZYNQ operating system.

The low-pass FIR filter designed in this study used 4479 logic elements, 1061 logic memory units, 6898 flip-flops, 2 RAM blocks, 29 DSP modules, and 1 buffer. Resource utilization report taken from VIVADO HLX is shown in Figure 6. When the energy consumption and temperature reports are examined, it is seen that the designed hardware consumes 1.431 W of energy. The junction temperature was calculated as 41.5 °C, and the thermal margin as 43.5 °C.

Resource	Utilization	Available	Utilization %
LUT	4479	53,200	8.42%
LUTRAM	1061	17400	6.10%
FF	6898	106,400	6.48%
BRAM	2	140	1.43%
DSP	29	220	13.18%
BUFG	1	32	3.13%

(a)



(b) (c)

Figure 6. (**a**) FPGA resource usage report. This report provides a comprehensive analysis of resource utilization for an FPGA implementation. (**b**) Resource usage graph. This graph presents a detailed analysis of the FPGA resource usage graph, illustrating the distribution of key hardware resources across the design. (**c**) FPGA power consumption graph. The logic module for the FIR filter consumes a total of 135 mW energy. The SoC Arm processor part on the PYNQ Z1 board consumes 1.296 W of energy.

The signal to be applied as an entrance to the system is obtained with the following Equation (2) using the numpy and matplotlib libraries in Python. The input signal generated is shown in Figure 7.

$$f(t) = 10{,}000\ x\ sin(0.2e6\ x\ 2\pi t) + 1500\ x\ cos(46e6\ x\ 2\pi t) + 2000\ x\ sin(12e6\ x\ 2\pi t) \quad (6)$$

The FIR filter design has been run on an Arm Cortex A9 CPU, a Tesla K80 GPU on Google Colab, and a PYNQ Z-1 board ZYNQ XC7Z020 FPGA. The following input signal was applied in three different systems, and the filtered signal on the side was obtained.

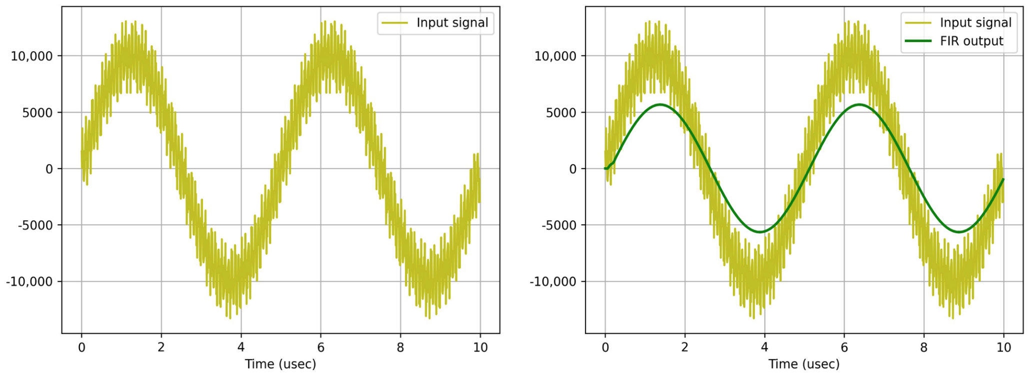


Figure 7. Behavior of two signals: (**left**) The noisy input signal generated from the equation. (**right**) The difference between the input and the filtered output signal depends on the filter's characteristics and the input signal's content.

While the software-based filtering process was performed using the CPU in 0.107 s, it was completed on the GPU in 0.008 s. The hardware-based filtering process on the FPGA was completed in just 0.004 s. These results indicate that the FPGA implementation is approximately 27 times faster than the CPU, while the GPU implementation is about 13 times faster than the CPU. Unlike CPUs, GPUs contain thousands of processing cores, allowing for true parallelism in computational tasks such as image processing or cryptocurrency mining. The architecture of FPGAs, however, is fundamentally different. Instead of relying on programmable cores or kernels, FPGAs implement parallelism at the hardware level through configurable logic blocks and interconnects. These logic blocks can be programmed to operate concurrently, enabling task-specific parallel data processing. While CPUs offer flexibility and are capable of executing a wide range of software with varying levels of complexity, GPU programming often requires adherence to architectural constraints and memory access patterns. In contrast, FPGA design involves creating dedicated digital circuits that can operate in parallel or series, depending on the application's requirements. These circuits are not software functions, but rather physical hardware connections configured to perform specific operations. As a result, the performance comparison outcomes of the filtering process are illustrated in Figure 8.

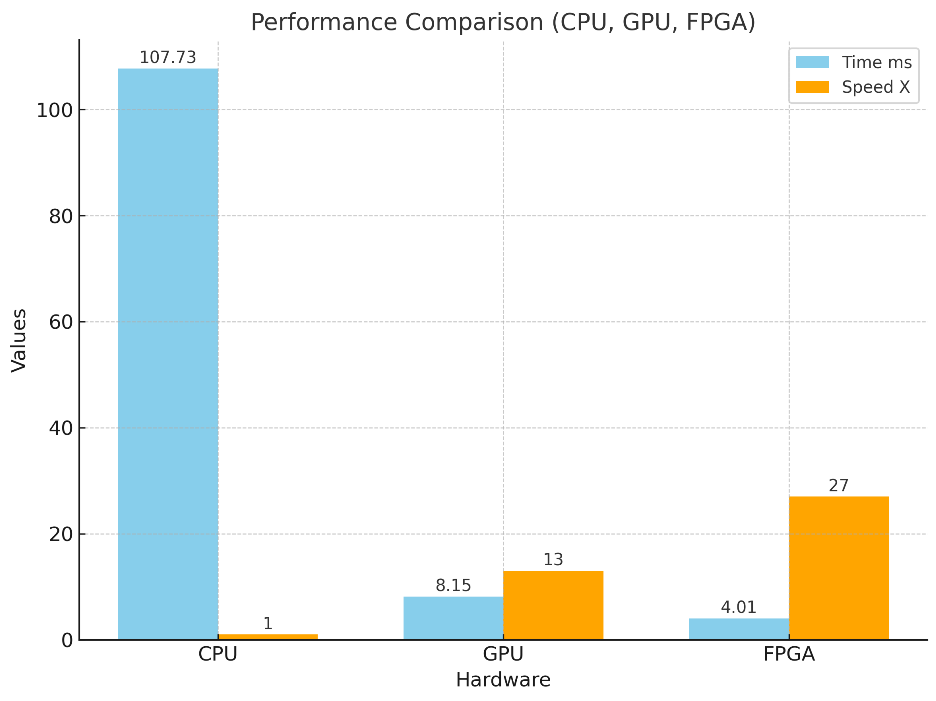


Figure 8. FIR filter process time analysis on CPU, GPU and FPGA. FIR filters are widely used in digital signal processing for their stability and linear-phase characteristics. Comparing their processing times across CPU, GPU, and FPGA platforms involves analyzing their performance in terms of computational efficiency, memory usage, and architecture-specific optimizations.

5. Discussion

This study compares the implementation of a finite impulse response (FIR) filter across CPU, GPU, and FPGA platforms. The CPU and FPGA implementations were integrated on a ZYNQ XC7Z020 chip using the AXI protocol, while the GPU implementation was executed on a Tesla K80 GPU via Google Colab, leveraging CUDA for parallel processing. The experimental results demonstrate that the FPGA outperforms the CPU by 27 times and the GPU by 2 times in processing speed (0.004 s, 0.107 s, and 0.008 s, respectively) while consuming lower energy at 1.431 W. The novelty of this study lies in its direct comparison of FPGA, GPU, and CPU platforms using a standardized FIR filter implementation, offering quantitative insights into their performance in a DSP context. Key contributions include the following: (1) demonstrating FPGA's 27 times and 2 times performance advantage over CPU and GPU, respectively, in FIR filter processing time (0.004 s for FPGA, 0.008 s for GPU, 0.107 s for CPU); (2) providing a comprehensive analysis of power consumption and resource utilization; and (3) offering practical guidance for selecting hardware platforms for real-time DSP applications in semiconductor-based systems.

Scalability is a critical factor in evaluating hardware platforms for FIR filter implementation. FPGAs offer scalability through their reconfigurable logic blocks, allowing parallel processing of multiple filter taps, as demonstrated by the use of 4479 logic elements and 29 DSP modules in this study. However, scalability is constrained by the available resources on the FPGA chip. GPUs, leveraging massive parallelism via CUDA on the

Tesla K80, excel in scaling to large datasets, making them suitable for high-throughput applications [17]. CPUs, with their sequential processing nature, exhibit limited scalability for compute-intensive tasks like FIR filtering, though they remain versatile for diverse workloads [18]. For larger FIR filters or datasets, FPGAs may maintain their performance advantage due to low-latency hardware customization, while GPUs could outperform in scenarios requiring extensive parallel computations.

The FIR filter implemented in this study, designed with a moderate number of taps using the Kaiser window method, represents a typical DSP task suitable for comparing computational efficiency across platforms. While the scale of the filter is relatively small, limiting its ability to fully stress the platforms' capabilities, the results (FPGA: 0.004 s, GPU: 0.008 s, CPU: 0.107 s) demonstrate FPGA's superior performance in low-latency applications. For larger FIR filters or datasets, GPUs may exhibit better scalability due to their parallel processing capabilities, as noted in [17]. Regarding energy efficiency, this study found that the FPGA consumed 1.431 W, comparable to or lower than the GPU's estimated 50–100 W on the Tesla K80. The literature suggests that FPGAs often outperform GPUs in energy efficiency for real-time DSP tasks due to their hardware-level customization and lack of operating system overhead [19,28]. Future work could explore larger FIR filters or alternative DSP tasks, such as IIR filters or FFT, to further validate these findings.

The implementation of the FIR filter across platforms utilized distinct programming tools, each with varying degrees of accessibility. For the FPGA, the VIVADO HLX suite, a proprietary tool, was used to design and synthesize the filter on the ZYNQ XC7Z020 chip, requiring expertise in hardware description languages (HDLs) like VHDL or Verilog. Open-source alternatives, such as Yosys or iCEstorm, exist for certain FPGA platforms but are less common for complex designs [31]. The GPU implementation leveraged NVIDIA's proprietary CUDA framework on the Tesla K80 via Google Colab, which, while accessible for parallel programming, requires knowledge of GPU-specific APIs. Open-source alternatives like OpenCL offer broader compatibility but similar complexity. The CPU implementation used Python with open-source libraries (NumPy, SciPy) on the PYNQ board's Linux OS, making it the most accessible due to Python's widespread use and simplicity [18]. Overall, CPU programming is the most user-friendly, followed by GPU, while FPGA programming demands specialized skills, impacting its adoption in non-specialist communities.

DSPs are specialized microprocessors optimized for signal processing tasks, offering efficient fixed-point arithmetic and multiply–accumulate operations ideal for FIR filter implementation [23]. Compared to FPGAs, DSPs provide lower development complexity but less flexibility in hardware customization. GPUs outperform DSPs in parallel processing for large datasets, while CPUs offer greater general-purpose flexibility but lower DSP-specific performance. In this study, DSPs were not included to focus on comparing general-purpose (CPU), parallel (GPU), and reconfigurable (FPGA) platforms, as these represent broader computational paradigms in semiconductor-based systems. However, DSPs could be considered in future work to evaluate their performance in FIR filtering, particularly for applications requiring low power and high efficiency in embedded systems.

Table 1 presents a comparative analysis of CPU, GPU, and FPGA platforms in the context of FIR filter implementation, evaluating latency, power consumption, timing performance, and time-to-market. This table integrates experimental results with findings from the literature to provide a more generalized reference.

Latency: FPGAs provide low latency through hardware-level customized logic blocks, offering a significant advantage in real-time signal processing tasks such as FIR filtering. In contrast, CPUs exhibit high latency due to their sequential processing architecture, while GPUs, despite their parallel processing capabilities, experience moderate latency due to fixed architectures. The FPGA's 0.004 s processing time in the FIR filter application demonstrates its suitability for latency-sensitive applications, such as real-time signal processing in IoT systems, as supported by prior studies [14].

Table 1. Comprehensive performance comparison of CPU, GPU, and FPGA.

Parameter	CPU	GPU	FPGA
Latency	High (sequential processing)	Moderate (parallel processing)	Low (custom hardware)
Power Consumption	∼5–10 W [25]	∼50–100 W [25]	1.431 W
Timing	0.107 s (FIR filter)	0.008 s (FIR filter)	0.004 s (FIR filter)
Time-to-Market	Short (easy programming)	Moderate (CUDA optimization)	Long (Verilog design)
Use Case	General-purpose computing	Deep learning	Real-time signal processing

Power Consumption: The FPGA's energy consumption of 1.431 W offers a substantial energy efficiency advantage compared to CPUs (estimated at ∼5–10 W) and GPUs (estimated at ∼50–100 W) [25]. This positions FPGAs as a preferred choice in energy-constrained scenarios, such as edge computing and battery-powered devices. However, the high power consumption of GPUs may be acceptable in large-scale parallel workloads, such as deep learning.

Timing: The measured processing times for the FIR filter (CPU: 0.107 s, GPU: 0.008 s, FPGA: 0.004 s) reflect the computational capabilities of each platform. The FPGA's superior performance stems from the hardware-level optimization of parallel processing pipelines. In contrast, the sequential processing of CPUs and the memory bandwidth constraints of GPUs limit their timing performance. The literature reports similar advantages for FPGAs in signal processing applications [13].

Time-to-Market: CPUs benefit from short development times due to high-level programming languages like Python/NumPy. GPUs require moderate development time, leveraging specialized libraries such as CUDA. FPGAs, designed using hardware description languages like Verilog and tools like Vivado HLx, entail longer development times. This poses a challenge to the widespread adoption of FPGAs, though application-specific optimizations may justify the cost.

The CPU implementation was performed on the Arm processor of the ZYNQ XC7Z020 chip, integrated into the PYNQ-Z1 board running a Linux operating system. The FIR filter algorithm was implemented using Python (https://www.python.org/) in a Jupyter Notebook environment, leveraging the NumPy and SciPy libraries for efficient signal processing computations. The implementation utilized single-threaded processing to maintain consistency with the sequential nature of CPU operations, resulting in a processing time of 0.107 s for the FIR filter benchmark.

In this study, we highlight the performance metrics of FIR filter implementations across FPGA, GPU, and CPU platforms, emphasizing FPGA's superior speed (27 times faster than CPU) and energy efficiency (30% less power consumption compared to GPU). These results underline the practical advantages of FPGA in real-time, low-latency applications such as autonomous driving, IoT, and medical devices, while also acknowledging the challenges of its steeper learning curve compared to GPU/CPU programming. A detailed analysis explains how FPGA's reconfigurable architecture enables efficient parallel processing, contrasting this with GPU and CPU strengths in programmability and general-purpose tasks. Furthermore, the findings are contextualized within broader industry trends, such as the demand for energy-efficient computing and the growth of edge computing. The discussion concludes by proposing future research directions, including the exploration of hybrid architectures and expanding the analysis to other use cases, to further advance the understanding and application of FPGA technology in diverse fields.

6. Conclusions

When designing an electronic device with a processor, the processing unit is initially selected in terms of speed, energy consumption, and processing capacity. Examples of these processing units are CPU, GPU, and FPGA processors, which offer several advantages in calculations. CPUs are the processors used in many devices, from computers to mobile phones, that can process sequential commands through input, output, and memory units. Due to sequential processing performed in CPUs, they are slower than GPUs and FPGAs in terms of repetitive intensive graphics and mathematical operations. They support many programming languages with the ability to work in hybrid with different processing units. GPUs have the highest bandwidth per pin. In particular, intensive mathematical operations can be performed at high speed. Since FPGAs are re-programmable logic gate arrays, pipeline operations can be designed. Modules that can work in parallel and serial on an FPGA can be designed. Owing to these capabilities, they are frequently employed in repetitive computational tasks, graphical data processing, and deep learning applications, where high efficiency and performance are critical. Various types of digital circuits can be built on an FPGA and loaded directly onto the board. In principle, everything can be calculated by a CPU, but other types of processors can be faster or more energy efficient with certain applications. In this study, FIR filter application was carried out on CPU, GPU, and FPGA. Power consumption and performance trends have been investigated, and FPGAs have been found to perform higher than GPUs and CPUs while consuming less energy.

This study provides a comprehensive comparison of FPGA, GPU, and CPU platforms in the implementation of a FIR filter, highlighting their performance in terms of processing time, power consumption, and resource utilization. Key contributions include demonstrating FPGA's significant performance advantage (0.004 s processing time, 27 times faster than CPU and 2 times faster than GPU), alongside comparable energy efficiency. These findings offer practical insights for selecting hardware platforms for real-time DSP applications in semiconductor-based systems. Future research directions include exploring larger FIR filter designs, alternative DSP tasks such as IIR filters or FFT, and the inclusion of DSPs to further evaluate specialized platforms. Additionally, hybrid architectures combining FPGA, GPU, and CPU strengths could be investigated to optimize performance for complex DSP applications.

The results of this study are strongly aligned with broader trends in the semiconductor and computing industries, which are increasingly driven by the demands of real-time processing, energy efficiency, and scalability. The demonstrated superiority of FPGA in terms of speed and energy consumption resonates with the rising adoption of edge computing, where low-latency and energy-efficient solutions are critical. As IoT continues to proliferate, devices deployed at the edge require lightweight, high-performance computing platforms, making an FPGA an ideal candidate for tasks such as data filtering, signal processing, and localized AI inference. In addition, the findings support the growing focus on sustainable technology in the semiconductor industry. With energy efficiency becoming a top priority due to environmental concerns and operational cost reduction, the low power consumption of FPGA systems positions them as a front-runner in sustainable computing solutions.

Future work can focus on extending the comparative analysis of FPGA, GPU, and CPU platforms to additional applications, such as image processing, cryptography, and deep learning, to assess their relative advantages in diverse computational tasks. Exploring hybrid architectures that integrate an FPGA with a GPU or a CPU could leverage the unique strengths of each platform, offering a promising solution for tasks that require both flexibility and high computational throughput. Furthermore, energy optimization techniques for FPGA implementations could be developed to further reduce power consumption, making them ideal for edge computing and battery-operated devices. Investigating advanced design automation tools to simplify FPGA programming would also enhance accessibility for developers. Finally, real-world deployments and dynamic reconfiguration studies could validate these findings in practical scenarios, allowing FPGA systems to adapt to changing workloads and expanding their application potential in industries such as healthcare, autonomous vehicles, and smart manufacturing.

Author Contributions: Conceptualization, M.A. and T.I.; methodology, T.I. and M.A.; software, M.A. and T.I.; validation, M.A. and T.I.; investigation, M.A. and T.I.; writing—original draft preparation, M.A. and T.I.; writing—review and editing, M.A. and T.I.; visualization, M.A. and T.I. All authors have read and agreed to the published version of the manuscript.

Funding: This study is partially financed by the European Union-NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project № BG-RRP-2.013-0001-C01.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bardeen, J.; Brattain, W. The transistor, a semi-conductor triode. *Phys. Rev.* **1948**, *74*, 230. [CrossRef]
2. Roup, R.; Kilby, J. Electrical Circuit Elements. U.S. Patent 2,841,508, 1958.
3. Schroder, D. *Semiconductor Material and Device Characterization*; John Wiley & Sons: Hoboken, NJ, USA, 2015.
4. El-Kareh, B.; Hutter, L. *Fundamentals of Semiconductor Processing Technology*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
5. Quirk, M.; Serda, J. *Semiconductor Manufacturing Technology*; Prentice Hall: Upper Saddle River, NJ, USA, 2001.
6. Bennett, B.; Magno, R.; Boos, J.; Kruppa, W.; Ancona, M. Antimonide-based compound semiconductors for electronic devices: A review. *Solid-State Electron.* **2005**, *49*, 1875–1895. [CrossRef]
7. Cardona, M.; Peter, Y. *Fundamentals of Semiconductors*; Springer: Berlin/Heidelberg, Germany, 2005.
8. Walia, K. Accelerating AI and Machine Learning in the Cloud: The Role of Semiconductor Technologies. *ESP Int. J. Adv. Comput. Technol.* **2024**, *2*, 34–41.
9. Arora, V.; Kossar, S.; Rasool, A.; Amiruddin, R.; Rasool, U.; Koser, K. Parallel Computation of Artificial Intelligence and Its Applications in Semiconductor Industry. In *Handbook Of Emerging Materials For Semiconductor Industry*; Springer: Singapore, 2024; p. 609.
10. Usamentiaga, R. CPU and GPU real-time filtering methods for dense surface metrology using general matrix to matrix multiplications. *J. Real-Time Image Process.* **2022**, *19*, 517–527. [CrossRef]
11. Mandelis, A.; Hess, P. *Semiconductors and Electronic Materials*; Spie Press: Bellingham, WA, USA, 2000.
12. Morkoç, H. *Handbook of Nitride Semiconductors and Devices, Materials Properties, Physics and Growth*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
13. Asano, S.; Maruyama, T.; Yamaguchi, Y. Performance comparison of FPGA, GPU and CPU in image processing. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 31 August 2009–2 September 2009; pp. 126–131.
14. Vestias, M.; Neto, H. Trends of CPU, GPU and FPGA for high-performance computing. In Proceedings of the 2014 24th International Conference On Field Programmable Logic And Applications (FPL), Munich, Germany, 2–4 September 2014; pp. 1–6.
15. Yeap, G. Smart mobile SoCs driving the semiconductor industry: Technology trend, challenges and opportunities. In Proceedings of the 2013 IEEE International Electron Devices Meeting, Washington, DC, USA, 9–11 December 2013; pp. 1–3.
16. Anwar, A.; Bhowmik, S.; Kadir, R.; Haque, M.; Rahman, S. Challenges in Implementing Machine Learning-Driven IoT Solutions in Semiconductor Design and Wireless Communication System. *Int. J. Recent Innov. Trends Comput. Commun.* **2024**, *12*, 872–889.
17. Ghafar-Zadeh, E.; Forouhi, S.; Azadmousavi, T. Advances in Electronic Biosensors. In *Advanced CMOS Biochips. Analog Circuits and Signal Processing*; Springer: Dordrecht, The Netherlands, 2024; pp. 197–224.
18. Jang, J.; Yoo, H. Review of Non-invasive, Wearable Biomedical Sensor and Imaging IC. In Proceedings of the 2024 IEEE Biomedical Circuits And Systems Conference (BioCAS), Xi'an, China, 24–26 October 2024; pp. 1–4.
19. Liu, Y.; Yu, Q.; Yang, L.; Cui, Y. Materials and biomedical applications of implantable electronic devices. *Adv. Mater. Technol.* **2023**, *8*, 2200853. [CrossRef]

20. Qiu, Z.; Shen, X.; Zhao, Z. Development Trends and Prospects of Semiconductor Devices and Technology. *Highlights Sci. Eng. Technol.* **2024**, *81*, 374–380. [CrossRef]
21. Liang, Y.; Zhao, C.; Yuan, H.; Chen, Y.; Zhang, W.; Huang, J.; Yu, D.; Liu, Y.; Titirici, M.; Chueh, Y. Others A review of rechargeable batteries for portable electronic devices. *InfoMat* **2019**, *1*, 6–32. [CrossRef]
22. Vaithianathan, M.; Patil, M.; Ng, S.; Udkar, S. Comparative Study of FPGA and GPU for High-Performance Computing and AI. *ESP Int. J. Adv. Comput. Technol. (ESP-IJACT)* **2023**, *1*, 37–46.
23. Zhang, S.; Wallscheid, O.; Porrmann, M. Machine learning for the control and monitoring of electric machine drives: Advances and trends. *IEEE Open J. Ind. Appl.* **2023**, *4*, 188–214. [CrossRef]
24. Mittal, S. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Comput. Appl.* **2020**, *32*, 1109–1139. [CrossRef]
25. Qasaimeh, M.; Denolf, K.; Lo, J.; Vissers, K.; Zambreno, J.; Jones, P. Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels. In Proceedings of the 2019 IEEE International Conference on Embedded Software and Systems (ICESS), Las Vegas, NV, USA, 2–3 June 2019; pp. 1–8.
26. Putnam, A.; Caulfield, A.; Chung, E.; Chiou, D.; Constantinides, K.; Demme, J.; Esmaeilzadeh, H.; Fowers, J.; Gopal, G.; Gray, J. Others A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Comput. Archit. News* **2014**, *42*, 13–24. [CrossRef]
27. Kocz, J.; Greenhill, L.; Barsdell, B.; Bernardi, G.; Jameson, A.; Clark, M.; Craig, J.; Price, D.; Taylor, G.; Schinzel, F. Others A Scalable Hybrid Fpga/gpu FX Correlator. *J. Astron. Instrum.* **2014**, *3*, 1450002. [CrossRef]
28. Vaithianathan, M. Real-Time Object Detection and Recognition in FPGA-Based Autonomous Driving Systems. *Int. J. Comput. Trends Technol.* **2024**, *72*, 145–152. [CrossRef]
29. Logeswaran, K.; Savitha, S.; Suresh, P.; Prasanna Kumar, K.; Gunasekar, M.; Rajadevi, R.; Dharani, M.; Jayasurya, A. Unifying Technologies in Industry 4.0: Harnessing the Synergy of Internet of Things, Big Data, Augmented Reality/Virtual Reality, and Blockchain Technologies. In *Topics In Artificial Intelligence Applied To Industry 4.0*; John Wiley Sons: Hoboken, NJ, USA, 2024; pp. 127–147.
30. Nurvitadhi, E.; Sheffield, D.; Sim, J.; Mishra, A.; Venkatesh, G.; Marr, D. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 International Conference On Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 77–84.
31. Che, S.; Li, J.; Sheaffer, J.; Skadron, K.; Lach, J. Accelerating compute-intensive applications with GPUs and FPGAs. In Proceedings of the 2008 Symposium on Application Specific Processors, Anaheim, CA, USA, 8–9 June 2008; pp. 101–107.
32. Litwin, L. FIR and IIR digital filters. *IEEE Potentials* **2000**, *19*, 28–31. [CrossRef]
33. Lim, Y.; Lee, J.; Chen, C.; Yang, R. A weighted least squares algorithm for quasi-equiripple FIR and IIR digital filter design. *IEEE Trans. Signal Process.* **1992**, *40*, 551–558. [CrossRef]
34. Karam, L.; McClellan, J. Chebyshev digital FIR filter design. *Signal Process* **1999**, *76*, 17–36. [CrossRef]
35. Isza, P. TFilter Web Application and FIR Filter Design Tool. 2025. Available online: http://t-filter.engineerjs.com/ (accessed on 12 May 2025).