ISSN: 2994-6433

Research Article

Journal of Sensor Networks and Data Communications

Real-Time Fusion Throughput on Spatial FPGA Fabric

Greg Passmore*

PassmoreLab, Austin, Texas, USA

***Corresponding Author**
Greg Passmore, PassmoreLab, Austin, Texas, USA.

Submitted: 2026, Jan 20; **Accepted**: 2026, Feb 27; **Published**: 2026, Mar 26

Citation: Passmore, G. (2026). Real-Time Fusion Throughput on Spatial FPGA Fabric. *J Sen Net Data Comm, 6*(2), 01-19.

Abstract

This paper provides a technical assessment of CPUs, GPUs, and FPGAs for real-time sensor fusion workloads involving radar, EO/IR imagery, LiDAR, inertial sensors, and RF data. The analysis covers throughput, latency, determinism, power efficiency, interconnect behavior, and support for large numbers of asynchronous sensor inputs. Empirical studies and published measurements show that CPUs offer flexibility but limited throughput and poor scaling under high-rate multi-sensor ingest. GPUs deliver the highest theoretical parallel throughput but depend on large batch sizes, incur millisecond-scale latency, and rely on PCIe transfers that constrain fusion performance. FPGAs provide deterministic, microsecond-scale processing, spatial parallelism for concurrent sensor pipelines, nanosecond-level timestamping, and significantly higher energy efficiency for structured DSP and streaming workloads. Benchmarks across FIR filtering, vision kernels, and real-time inference demonstrate FPGA speedups ranging from modest to substantial relative to CPUs and GPUs, with consistent gains in throughput per watt. FPGAs also mitigate interconnect bottlenecks by performing in-situ reduction before data reaches GPUs or CPUs. My analysis indicates that heterogeneous architectures, with FPGAs as front-end fusion engines and GPUs and CPUs as downstream analytic and control elements, provide the most effective performance profile for modern sensor fusion systems.

Keywords: Sensor Fusion, FPGA Acceleration, Deterministic Latency, Heterogeneous Computing, Throughput Per Watt

1. Introduction

Sensor fusion applications such as combining radar (RF) with electro-optical/infrared (EO/IR) imagery, or fusing LiDAR, cameras, and other sensors, demand high data throughput and strict real-time performance. Multiple asynchronous sensor streams must be processed and merged with minimal delay to enable accurate perception and tracking. This paper compares Field-Programmable Gate Arrays (FPGAs), Central Processing Units (CPUs), and Graphics Processing Units (GPUs) for sensor fusion workloads, focusing on throughput, latency and jitter, determinism, parallelism, power efficiency, I/O constraints, and multi-sensor handling. Quantitative performance indicators and example use cases (RF+EO/IR fusion, radar+vision processing, real-time target tracking) are drawn from documented FPGA-based sensor systems, comparative CPU/GPU/FPGA benchmarks, and large-scale FPGA deployments in production systems [1-6]. The performance advantages of FPGAs over CPUs and GPUs in representative fusion workloads while preserving a technical, analytical style grounded in the cited literature.

2. Background and Historical Context

Ross Freeman (1948–1989) was an American electrical engineer who co-founded Xilinx in 1984 alongside Bernard Vonderschmitt and James Barnett. Freeman invented the field-programmable gate array, securing the foundational patent (U.S. Patent 4,870,302, filed 1984, granted 1989) and bringing the first commercial FPGA to market that same year. The core idea, a programmable logic fabric whose connections and functions could be configured after fabrication by the end user, represented a fundamental departure from the fixed-function ASICs and simple programmable logic devices that preceded it. Freeman was inducted into the National Inventors Hall of Fame in 2009, two decades after his death at age 41. I find it remarkable that the architecture he conceived in the mid-1980s, originally envisioned for glue logic and simple control tasks, now underlies gigahertz-clocked accelerators with thousands of hardened DSP slices,

AI engines, and multi-gigabit serial transceivers. The distance from the XC2064 to a Versal AI Core is extraordinary.

The evolution from early FPGAs to modern adaptive SoC fabrics spans four decades of expanding device density, embedded hard-IP blocks, and increasingly capable on-chip interconnects. Early FPGAs were modest in scale; subsequent generations added embedded block RAM, then multiplier–accumulator (DSP) slices optimized for fixed-point signal processing, then hard-IP PCIe, Ethernet, and memory controllers, and most recently AI engine arrays with dedicated matrix-multiply datapaths. The Xilinx Versal family, for example, integrates programmable logic, embedded ARM cores, and a dedicated AI engine array on a single device, connected through an internal network-on-chip [7]. This progression is directly relevant to sensor fusion: each generation of improvements expanded the class of workloads an FPGA could handle cost-effectively, moving from simple glue logic to full radar and imaging pipelines.

Andrew Putnam is a key figure in translating FPGA potential into production-scale deployment. As the principal architect of Microsoft's Project Catapult, Putnam and his collaborators at Microsoft Research demonstrated in 2014 that FPGAs could be deployed at hyperscale, meaning every server in a production datacenter fleet, to accelerate search ranking and other large-scale services [3]. This was the first demonstration at that scale that FPGAs were viable in the cloud datacenter context, not merely in embedded or defense applications. Putnam is now a Partner in Microsoft's Azure Hardware Engineering group, where FPGA-based acceleration continues to be a core part of Azure infrastructure. The Catapult architecture, which places an FPGA between the CPU and the network, is structurally analogous to what this paper recommends for sensor fusion: an FPGA front end that preprocesses and reduces data before it reaches the CPU or GPU back end.

The broader intellectual context. Claude Shannon (1916–2001), an American mathematician and electrical engineer at Bell Labs and MIT, established information theory and the mathematical foundations of digital circuit design in the late 1940s. Shannon's work on switching circuits formalized the relationship between Boolean algebra and physical logic gates, making systematic digital design possible. Every FPGA lookup table, every logic optimization pass in modern synthesis tools, and every pipelined DSP core traces its theoretical lineage to Shannon's circuit analysis framework. His 1948 paper on the mathematical theory of communication is the source of the bit as a unit of information; his 1937 master's thesis is arguably the founding document of digital design.

The GPU computing revolution is the other major context for this comparison. NVIDIA's CUDA platform, introduced in 2007, democratized general-purpose GPU programming and initiated a period of rapid growth in GPU-accelerated workloads. By the early 2010s, GPUs had become the dominant platform for deep neural network training, and by the mid-2010s they were widely used for inference in datacenters. GPUs excel at dense, regular parallel computation, particularly matrix multiplications and convolutions, which map perfectly to their architecture of thousands of identical cores executing the same instruction stream. Their limitations in the real-time sensor fusion context, batching requirements, PCIe dependency, and non-deterministic latency, are architectural consequences of this design philosophy, not incidental deficiencies, and I address them quantitatively in the sections that follow.

3. Throughput and Parallelism

Throughput is measured in operations or samples per second. GPUs excel at raw parallel throughput: modern accelerators such as the NVIDIA A100 sustain on the order of 10^4 low-precision operations per second (hundreds of tera-operations per second for FP16 or INT8) when heavily loaded with dense linear algebra or convolutional kernels [8-10]. CPUs, by contrast, have a small number of general-purpose cores. Even with wide vector instruction sets (AVX2, AVX-512) and multicore operation, sustained throughput for real-world stencil and signal processing workloads is typically closer to 10^9 to 10^{10} effective operations per second once memory bandwidth and cache behavior are considered [4,10].

FPGAs occupy an intermediate and more flexible position. Although clocked at lower frequencies than CPUs or GPUs, they can implement deeply pipelined, wide datapaths and multiple replicated pipelines configured exactly to the algorithm. In streaming sensor workloads, an FPGA design that accepts one sample per clock per pipeline can sustain aggregate throughputs in the 10^{11} to 10^{13} operations-per-second range on mid- to high-end devices, particularly when arithmetic is mapped to dedicated DSP slices or AI engines [3,1,2,7]. The key property is that, for a fixed sensor pipeline, almost every clock cycle processes a new sample rather than waiting on instruction dispatch or thread scheduling.

The distinguish between peak theoretical throughput and effective throughput on streaming sensor workloads. GPU peak numbers assume very high occupancy with tens of thousands of concurrent threads executing nearly identical kernels on large batches of data. In practice, if an algorithm or sensor fusion task does not naturally expose that level of data parallelism, or if performance is dominated by I/O and pre-processing, effective GPU throughput can drop well below the theoretical peak [9,4]. CPUs face similar challenges; they cannot easily hide DRAM latency or context-switch overhead when interacting with many concurrent sensor streams.

FPGAs, by contrast, can tailor hardware directly to the streaming data. Data moves through a fixed custom pipeline, and every clock cycle performs useful work on a new sample, provided the design is properly pipelined. A recent FIR filter case study found that an FPGA implementation executed a representative DSP workload about 27 times faster than a CPU and roughly twice as fast as a GPU when both were optimized for the same single-input stream; measured execution times were approximately 0.004 s for the FPGA, 0.008 s for the GPU, and 0.107 s for the CPU on the same filter configuration [5]. This is consistent with earlier comparative studies of image-processing kernels, which reported that for certain 2D vision filters, FPGA implementations achieved 1.2× to more than 20× better energy efficiency than CPU and GPU implementations while maintaining equal or higher frame rates [4].

High-energy physics trigger-processing studies provide another example where workload structure resembles streaming sensor fusion. A recent comparison of FPGA and GPU implementations of a machine-learning-based track reconstruction pipeline at LHCb found that FPGAs achieved similar event throughput to GPUs but with significantly lower power consumption, yielding higher throughput per watt for tightly pipelined inference workloads [6]. Large-scale datacenter deployments of FPGA fabrics demonstrate that FPGAs can provide competitive throughput to CPU-only services while improving energy efficiency and latency at production scale [3].

Parallelism and concurrency also differ fundamentally across the three architectures. CPUs execute mostly sequential instruction streams with limited thread-level and SIMD parallelism. GPUs exploit massive data parallelism across many identical cores but operate most efficiently when all cores execute the same kernel on different data in lock-step fashion. When multiple dissimilar tasks or many sensor streams must be processed concurrently, GPU resources must be time-shared, with an associated overhead in scheduling, kernel launch, and context management.

FPGAs implement explicit parallelism by instantiating separate hardware blocks for each task or sensor stream. If the system must handle N sensors, each with its own chain of operations, the FPGA can host N independent pipelines, each running continuously. Surveys of FPGA-based sensor systems emphasize that this spatial parallelism is a primary reason FPGAs are attractive for multi-sensor data acquisition and preprocessing: each sensor path can be mapped to a dedicated hardware pipeline with predictable throughput and without interference from other paths [1,2]. This property makes FPGAs particularly well suited to multi-sensor fusion workloads where multiple streams must be processed concurrently and then fused.

4. Latency, Jitter, and Determinism

Latency and jitter are central performance metrics for real-time sensor fusion. Latency denotes the time between sensor input and fused output. Jitter denotes the variability of this delay across frames or events. FPGAs are well established as providing the lowest and most deterministic latency among the three architectures due to their synchronous, spatially pipelined execution model.

In an FPGA, each stage of the processing pipeline corresponds to a fixed-depth register stage with fully deterministic propagation delay. Once synthesized, the number of cycles required to process a sample is constant, and cycle-accurate timing is guaranteed except for clock-level variation. End-to-end latencies of a few microseconds for nontrivial DSP pipelines are routinely reported in embedded and data-acquisition systems [1,2]. In time-distribution and synchronization networks, FPGA-based IEEE-1588 and related timing engines have achieved nanosecond-scale synchronization accuracy across thousands of distributed nodes, demonstrating that FPGA timestamping and processing paths can be bounded extremely tightly [11-13].

In contrast, CPUs executing under a general-purpose operating system typically exhibit latencies from tens of microseconds to milliseconds for sensor-processing pipelines, along with significant jitter caused by scheduler preemption, interrupt handling, cache misses, and memory contention. Even with real-time kernels, the sequential instruction model and shared memory hierarchy of CPUs introduces variability that is difficult to constrain. Embedded sensor-system surveys consistently identify jitter originating from OS activity as a primary limitation when CPUs are used in tight control loops or for hard real-time synchronization [1].

Discrete GPUs fall between CPUs and FPGAs with respect to latency but are not suitable for hard real-time deadlines. Data must be transferred to GPU memory, typically across PCIe, and GPU kernels must be launched, which incurs overhead even in the best-case scenario. The requirement for large batch sizes to reach peak occupancy introduces further latency, because initial samples must wait until a sufficient batch is collected. Measurements in comparative GPU/FPGA studies show that GPU pipeline latency typically lies in the hundreds of microseconds to millisecond range, with additional variability from kernel scheduling and device contention [4,6]. While GPUs provide exceptional average throughput, their latency variance makes them unsuitable for deterministic low-level fusion, time alignment, or control-loop operation.

Determinism is the area where FPGAs hold the clearest advantage. Because FPGA pipelines do not rely on instruction issue, thread

scheduling, or cache hierarchies, the worst-case execution time is effectively identical to the nominal execution time for a given design. Jitter can be reduced to tens of nanoseconds or less, as demonstrated in FPGA-based timing networks achieving 10–20 ns cross-node stability [11,12]. In multi-sensor navigation systems, FPGA-based timestamping proves crucial in achieving sub-millisecond fusion accuracy between camera, IMU, GNSS, and other sensors [14].

In summary, for real-time fusion tasks requiring bounded latency, low jitter, and precise time alignment between heterogeneous sensor streams: FPGAs provide deterministic microsecond-scale latency; GPUs provide high throughput but millisecond-scale variable latency; and CPUs exhibit the largest latency variability due to OS behavior and memory contention. This ordering is well-documented across embedded systems, distributed timing architectures, and data-acquisition pipelines, and directly informs hardware selection in time-critical fusion systems.

5. Mathematical Model

5.1. CPU Latency Model

For a CPU-based fusion pipeline, the end-to-end latency from sensor input to fused output can be written as

$$L_{CPU} = L_{ingest} + L_{queue} + L_{sched} + L_{proc}$$

Here L_{ingest} is the sensor-ingest and driver overhead, L_{queue} is queuing delay in software buffers, L_{sched} is latency added by OS scheduling and context switching, and L_{proc} is the pure computation time of the fusion algorithm on the CPU cores.

If the CPU is modeled as an $M/M/1$-like server with mean service time S_{CPU} at utilization ρ_{CPU} , the expected sojourn time of a task is approximated as

$$L_{CPU} \approx \frac{S_{CPU}}{1 - \rho_{CPU}} + L_{ingest} + L_{sched}$$

where S_{CPU} includes both application processing and cache/memory penalties, and L_{sched} captures additional scheduler and interrupt overhead that is not well modeled by simple queueing.

Latency jitter for the CPU can be represented by the variance of the total latency

$$\sigma^2_{CPU} = \mathrm{Var}(L_{ingest}) + \mathrm{Var}(L_{queue}) + \mathrm{Var}(L_{sched}) + \mathrm{Var}(L_{proc})$$

The dominant contributions typically come from $\mathrm{Var}(L_{sched})$ and $\mathrm{Var}(L_{queue})$, reflecting OS preemption, cache misses, and contention on shared resources.

6. GPU Latency Model

For a discrete GPU used as an accelerator, the latency from sensor input (at the host or FPGA front end) to fused output on the GPU can be decomposed as

$$L_{GPU} = L_{in,PCIe} + L_{batch} + L_{launch} + L_{kernel} + L_{out,PCIe}$$

Here $L_{in,PCIe}$ is the time to transfer input data from sensor or host memory into GPU memory, L_{batch} is the time a sample waits while a batch is accumulated to reach sufficient occupancy, L_{launch} is the kernel launch and scheduling overhead, L_{kernel} is the execution time of the GPU kernel, and $L_{out,PCIe}$ is the time to transfer fused results back to the host or to another device.

If a batch of size B samples is processed in a kernel that runs for time $T_{kernel}(B)$, then the per-sample processing component can be approximated as

$$L_{kernel,per\,sample}(B) \approx \frac{T_{kernel}(B)}{B}$$

Assuming PCIe bandwidth R_{PCIe} and per-batch payload sizes D_{in} and D_{out}, the PCIe contributions can be written as

$$L_{in,PCIe} \approx \frac{D_{in}}{R_{PCIe}}$$

$$L_{out,PCIe} \approx \frac{D_{out}}{R_{PCIe}}$$

The total expected latency for a typical sample in a steady-state GPU pipeline can then be approximated as

$$L_{GPU} \approx \frac{D_{in}}{R_{PCIe}} + L_{batch} + L_{launch} + \frac{T_{kernel}(B)}{B} + \frac{D_{out}}{R_{PCIe}}$$

Latency jitter on the GPU can be modeled by the variance of these components

$$\sigma^2_{GPU} = \mathrm{Var}(L_{in,PCIe}) + \mathrm{Var}(L_{batch}) + \mathrm{Var}(L_{launch}) + \mathrm{Var}(L_{kernel}) + \mathrm{Var}(L_{out,PCIe})$$

Here Var(L_{batch}) and Var(L_{launch}) capture variability from batching and scheduling, and Var($L_{in,PCIe}$) and Var($L_{out,PCIe}$) capture contention on the interconnect.

7. FPGA Latency Model

For an FPGA-based streaming fusion pipeline, the latency is determined by the number of pipeline stages and the clock period. If the pipeline from sensor input to fused output has depth D (in clock cycles) and the FPGA clock period is T_{clk}, then the core pipeline latency is

$$L_{pipe} = D, T_{clk}$$

If there is an additional fixed interface latency L_{iface} for serializer, deserializer, or ADC/DAC crossings, the total FPGA latency can be written as

$$L_{FPGA} = L_{iface} + D, T_{clk}$$

In a steady streaming regime, each new sample enters the pipeline every clock cycle. The throughput is $1/T_{clk}$ samples per second per pipeline, while the latency is fixed at D, T_{clk} independent of load, as long as backpressure does not occur.

Because every path through the pipeline uses the same number of registers and combinational stages, the variance of FPGA latency in the absence of backpressure is effectively zero at the design level

$$\sigma^2_{FPGA} \approx 0$$

Any residual jitter is dominated by clock variation, In practice, residual jitter includes clock jitter, metastability in I/0 domain crossings, and backpressure events. and external interface effects, which are typically orders of magnitude smaller than the variability observed on CPUs and GPUs.

8. Comparative Latency Summary

Using these models, the qualitative ordering can be summarized as

$$L_{FPGA} \approx L_{iface} + D, T_{clk}$$

$$L_{GPU} \approx \frac{D_{in}}{R_{PCIe}} + L_{batch} + L_{launch} + \frac{T_{kernel}(B)}{B} + \frac{D_{out}}{R_{PCIe}}$$

$$L_{CPU} \approx \frac{S_{CPU}}{1 - \rho_{CPU}} + L_{ingest} + L_{sched}$$

with jitter levels

$$\sigma^2_{FPGA} \ll \sigma^2_{GPU} \approx \sigma^2_{CPU}$$

which captures the deterministic nature of FPGA pipelines, the batched and PCIe-bound behavior of GPUs, and the scheduler- and contention-driven variability of CPUs.

9. Power Efficiency and Throughput per Watt

Power efficiency is a defining constraint in deployed sensor fusion systems, particularly in airborne, ground-vehicle, and battery-powered environments. Throughput per watt is a central comparison metric between CPUs, GPUs, and FPGAs. Across a substantial range of documented workloads, including FIR filtering, image-processing kernels, database acceleration, and real-time inference, FPGAs typically offer higher energy efficiency than GPUs and significantly higher efficiency than CPUs.

A large-scale datacenter deployment demonstrated this effect at production scale. In Microsoft's Catapult project, a reconfigurable FPGA fabric achieved up to several-fold improvements in performance per watt relative to CPU-only implementations for several large-scale services, including ranking, image processing, and search [3]. Although those tasks differ from sensor fusion workloads, they exhibit the same structured streaming properties: fixed pipelines, high data rates, and low tolerance for irregular control flow.

More recent comparative studies provide direct quantitative measurements for DSP and sensor-like workloads. In a FIR filter benchmark, a representative FPGA implementation completed a fixed filtering task in approximately 0.004 s while consuming roughly 1.4 W, compared to 0.008 s at substantially higher GPU power and 0.107 s at still higher CPU power [5]. Because the FPGA both executed twice as fast as the GPU and consumed markedly less power, its throughput per watt exceeded that of the GPU by a large margin. Energy per operation was lower by roughly a factor of two to three compared to the GPU and far lower compared to the CPU.

Vision-kernel benchmarks show similar trends. In a study comparing CPU, GPU, and FPGA implementations of 2D image-processing filters, the FPGA achieved 1.2× to more than 20× higher energy efficiency than the CPU and GPU, depending on the kernel and device configuration [4]. These results were obtained on representative convolution, filtering, and aggregation kernels typical of the pre-processing stages in EO/IR fusion pipelines.

Workloads in high-energy physics provide a close analogue to real-time multi-sensor fusion, because they involve large data streams, fixed-latency constraints, and online filtering. A recent comparison showed that an FPGA-based track-reconstruction inference pipeline achieved similar event-rate throughput to a GPU pipeline while consuming significantly less energy per processed event, yielding a superior throughput-per-watt ratio even at equal computational throughput [6].

On modern AI workloads, GPUs remain extremely high-throughput accelerators, especially when operating at batch sizes that maximize core occupancy. However, on strictly streaming applications with small or fixed batch sizes, such as continuous multi-sensor ingestion, low-latency CNN pipelines used in robotics, and online inference inside a fusion loop, FPGAs often demonstrate higher effective energy efficiency. Comparative surveys of GPU vs FPGA implementations for embedded intelligence and robotics report that FPGA-based accelerators achieve substantially lower energy per inference when operating in real-time, batch-size-1 or batch-size-few modes [4,15].

CPUs generally exhibit the lowest energy efficiency for numerically intensive workloads. Even when vector extensions are used, the CPU power budget must support branch logic, memory hierarchies, general-purpose execution, and OS overhead. For structured DSP or fusion pipelines, CPU energy per processed sample is typically orders of magnitude higher than that of an FPGA, and still significantly higher than the GPU for highly parallel, compute-bound tasks [4,5].

In energy-constrained environments, such as unmanned aerial vehicles, autonomous ground units, or satellite payloads, these differences are decisive. FPGA-based fusion pipelines can be dimensioned so that only the required logic toggles each clock cycle. This enables fixed power envelopes and stable thermal behavior while still meeting throughput requirements. GPU-based pipelines require large portions of the device to remain active even when many cores are not fully utilized, and CPU-based pipelines typically expend tens of watts to maintain real-time rates on high-bandwidth sensor setups. The combined experimental evidence across DSP, vision, inference, and large-scale service workloads consistently shows that FPGAs offer superior throughput per watt for structured streaming tasks

relevant to sensor fusion.

10. Definition of Throughput per Watt

For any platform, define

$$\eta = \frac{\Phi}{P}$$

where Φ is sustained throughput (operations per second, samples per second, or tasks per second under a defined workload) and P is average electrical power (watts) during steady-state operation. The unit of η is operations per joule or samples per joule

11. CPU Throughput per Watt

Let Ncores be the number of CPU cores, v the vector width in operations per cycle, f_{CPU} the clock frequency, and α_{CPU} an efficiency factor describing how well the workload uses available compute (including cache, memory bandwidth, and branch behavior):

$$\Phi_{CPU} \approx \alpha_{CPU}, N_{cores}, v, f_{CPU}$$

If the CPU package power during the workload is P_{CPU}, the effective throughput per watt is

$$\eta_{CPU} = \frac{\Phi_{CPU}}{P_{CPU}} \approx \frac{\alpha_{CPU}, N_{cores}, v, f_{CPU}}{P_{CPU}}$$

For streaming sensor fusion workloads, α_{CPU} is often small due to cache misses, DRAM latency, and OS overhead, which reduces η_{CPU} even when N_{cores} and v are large.

12. GPU Throughput per Watt

Let NSM be the number of streaming multiprocessors, $N_{cores,SM}$ the number of cores per SM, f_{GPU} the GPU core clock, and α_{GPU} the occupancy and efficiency factor for the workload (how well it fills warps, hides latency, and uses the memory system):

$$\Phi_{GPU} \approx \alpha_{GPU}, N_{SM}, N_{cores,SM}, f_{GPU}$$

If the GPU power under load is P_{GPU}, then

$$\eta_{GPU} = \frac{\Phi_{GPU}}{P_{GPU}} \approx \frac{\alpha_{GPU}, N_{SM}, N_{cores,SM}, f_{GPU}}{P_{GPU}}$$

For dense linear algebra and large-batch inference, α_{GPU} can approach 1, giving very high raw η_{GPU}. For small-batch or strictly streaming fusion workloads, α_{GPU} is typically reduced by underutilization of cores, memory-bound behavior, and idle periods between kernel launches, lowering η_{GPU} relative to peak specifications.

13. FPGA Throughput per Watt

For an FPGA-based streaming pipeline, define N_{pipe} as the number of parallel pipelines instantiated, f_{FPGA} as the clock frequency, and c as the average number of useful operations performed per sample within a single pipeline.

Assuming one new sample per clock per pipeline in steady state:

$$\Phi_{FPGA} \approx N_{pipe}, c, f_{FPGA}$$

If the FPGA dynamic power during the workload is P_{FPGA}, then

$$\eta_{FPGA} = \frac{\Phi_{FPGA}}{P_{FPGA}} \approx \frac{N_{pipe}, c, f_{FPGA}}{P_{FPGA}}$$

Because only the logic implementing the pipelines is toggling, and there is no general-purpose instruction overhead, N_{pipe} and c directly reflect useful work. For structured DSP and fusion workloads, this leads to high ηFPGA even at modest f_{FPGA} and moderate P_{FPGA}.

14. Relative Comparison

To compare architectures under the same workload, fix the required sustained throughput Φ_{target} and solve for the power each platform must expend:

CPU:

$$P_{CPU} \approx \frac{\alpha_{CPU}, N_{cores}, v, f_{CPU}}{\eta_{CPU}} = \frac{\Phi_{target}}{\eta_{CPU}}$$

GPU:

$$P_{GPU} \approx \frac{\Phi_{target}}{\eta_{GPU}}$$

FPGA:

$$P_{FPGA} \approx \frac{\Phi_{target}}{\eta_{FPGA}}$$

Experimental results from FIR filtering and vision kernels can be summarized as

$$\eta_{FPGA} \approx k_1, \eta_{GPU} \approx k_2, \eta_{CPU}$$

with k_1 typically in the range 2 to 10 and k_2 significantly larger, consistent with reported factors of 2– 10 better energy efficiency than GPUs and substantially more than CPUs for structured streaming workloads. Inserting these into the expressions above directly yields lower P_{FPGA} for the same Φ_{target}, which matches the empirical observations cited in the paper.

Example Numeric Comparison for Throughput per Watt

The following illustrative example uses representative values derived from reported ranges, not measured values from a single experiment. Assume a target sustained throughput of 5.0×10^{11} operations per second (500 Gops/s) for a representative streaming fusion workload.

Platform Sustained throughput (ops/s) Throughput per watt (ops/J) Required power (W)

Platform	Throughput (ops/s)	Throughput per watt (ops/J)	Required power (W)
CPU	5.0e11	1.0e9	500
GPU	5.0e11	8.0e9	62.5
FPGA	5.0e11	2.0e10	25

In this example, the CPU achieves an effective 1.0×10^9 operations per joule, the GPU achieves 8.0×10^9 operations per joule, and the FPGA achieves 2.0×10^{10} operations per joule on the same class of streaming workload. To sustain the same 5.0×10^{11} ops/s:

- The CPU must operate at approximately 500 W.
- The GPU can meet the target at about 62.5 W.
- The FPGA can meet the target at about 25 W.

This numeric instance directly reflects the algebraic relations

$$\eta = \frac{\Phi}{P}$$

$$P = \frac{\Phi}{\eta}$$

and encodes an FPGA efficiency advantage of 2.5× over the GPU and 20× over the CPU for this representative sensor-fusion pipeline, consistent with the ranges reported in the cited FIR and vision-kernel studies.

15. Handling Multiple Asynchronous Sensor Inputs

Let's turn to the ingestion problem. Sensor fusion requires ingesting heterogeneous sensor streams that often operate at different rates, different resolutions, and with different timing characteristics. The fusion engine must capture, preprocess, time-align, and combine these streams in real time. FPGAs are particularly well suited to this because they can implement multiple sensor interfaces and customizable streaming pipelines in parallel hardware.

FPGAs expose rich I/O capabilities, including multi-gigabit serial transceivers, LVDS/MIPI interfaces for cameras, JESD204B/C for high-speed ADCs, and parallel interfaces for structured sensor buses. These interfaces can be instantiated simultaneously, enabling direct attachment to cameras, LiDAR systems, radar front ends, GNSS receivers, and inertial sensors. Surveys of FPGA-based embedded sensor systems document widespread use of FPGAs as multi-sensor front ends, emphasizing their ability to attach directly to heterogeneous sensors and service each stream with predictable latency and throughput [1,2].

Each sensor channel can be mapped to a dedicated processing pipeline that performs tasks such as format conversion, digital filtering, buffering, decimation, and metadata annotation. Because these pipelines operate in spatial parallelism, adding a sensor does not require time-sharing existing compute resources. Instead, new hardware is instantiated, subject to logic and DSP availability, allowing each sensor to run continuously and independently. In contrast, CPUs must service each sensor through threads, interrupts, or driver routines, which compete for shared execution resources and can introduce latency and jitter [1].

A critical component of multi-sensor fusion is timestamping and synchronization. FPGAs can attach hardware-level timestamps to incoming data using counters disciplined by high-stability oscillators or network-distributed time bases. In IEEE 1588-based synchronization architectures, FPGA NICs and PHYs can perform timestamping at the MAC or PHY layer, achieving nanosecond-level precision across distributed systems [11]. Enhanced FPGA-based timing systems demonstrate that comparable nanosecond-level alignment can be maintained over long durations with minimal drift [12,13]. This deterministic timestamping is essential for fusing radar, EO/IR, LiDAR, and inertial data where sub-millisecond alignment affects track accuracy.

CPUs and GPUs can also timestamp data, but typically do so at higher protocol layers or in software. These timestamps are subject to jitter arising from interrupt latency, driver behavior, and OS scheduling, and cannot reliably capture the instant of physical sampling. In high-speed systems, these uncertainties accumulate and degrade the quality of time alignment.

Representative multi-sensor systems demonstrate how FPGA-based ingestion enables consistent fusion performance. A GNSS–visual–inertial navigation suite integrated an FPGA for sensor synchronization and precise timing distribution, achieving sub-millisecond fusion accuracy and stable alignment between three heterogeneous sensor modalities [14]. FPGA-based radar front ends similarly perform digital downconversion, channelization, Doppler processing, and thresholding in streaming mode while separate logic regions handle video capture or image pre-processing, with all outputs timestamped using a shared hardware time base. Without this separation, a CPU would be forced to ingest raw RF streams and video data simultaneously, perform all low-level DSP and buffer management in software, and attempt to maintain sub-millisecond alignment; a task that typically exceeds practical CPU capacity at high sensor rates.

To recap, FPGAs are well adapted to handling multiple asynchronous sensor inputs because they provide:
- direct high-speed interfaces to heterogeneous sensors,
- dedicated per-sensor pipelines with deterministic timing,
- hardware timestamping and synchronization at nanosecond precision, and
- predictable, low-latency behavior even as sensor count increases.

CPUs and GPUs, by contrast, must rely on software for ingestion and synchronization, making them more susceptible to jitter, contention, and bandwidth bottlenecks under multi-sensor load.

16. Impact of PCIe and Interconnect Bottlenecks

In our work, PCIe is the recurring bottleneck we keep running into. When GPUs or discrete FPGAs operate as accelerators in a sensor fusion system, the characteristics of the interconnect, typically PCI Express (PCIe), become a major determinant of overall throughput and latency. Sensor data often arrives at an embedded processor or FPGA connected to the physical sensors, and must then be transferred across PCIe to a GPU for high-level processing. The bandwidth, latency, and protocol behavior of this interconnect directly influence

achievable fusion rates, especially in multi-sensor configurations.

PCIe Gen3 ×16 provides a theoretical maximum of approximately 15.75 GB/s and PCIe Gen4 ×16 approximately 31.5 GB/s, assuming ideal conditions. Real-world throughput is lower due to packet overhead, credit-based flow control, and platform-level topology. High-resolution EO/IR cameras, LiDAR systems, and wideband RF capture devices can each produce hundreds of megabytes per second, so several such sensors operating simultaneously can saturate one or more PCIe links [9,16].

Traditional CPU-to-GPU transfer paths require memcpy operations between device buffers, host memory, and GPU memory. This adds latency and consumes CPU cycles. Techniques such as NVIDIA GPUDirect RDMA eliminate the intermediate host-memory copy by allowing NICs and FPGAs to DMA directly into GPU memory, significantly reducing latency and CPU load. Experimental evaluations demonstrate that GPUDirect RDMA can approach the effective line rate of PCIe (multiple GB/s) and reduce transfer latency from millisecond-scale to microseconds for appropriately sized buffers [16,9]. Despite this improvement, the aggregate bandwidth of the PCIe fabric remains a limiting factor in multi-sensor systems, especially where multiple GPUs and multiple high-rate sensors share the same PCIe hierarchy.

In complex sensor fusion systems, PCIe switching adds additional constraints. Peer-to-peer transfers between GPUs or between a GPU and a discrete FPGA may require routing through a PCIe switch or host bridge, depending on topology. This introduces additional hop latency, reduces available bandwidth per path, and can result in nondeterministic arbitration delays. Standard practice in embedded sensor design emphasizes that placing FPGA logic as close as possible to the data sources, often within the same device as the data converters, significantly reduces interconnect traffic and improves determinism (de la Piedra, Braeken, and Touhafi 2012; Garcia et al. 2014). This architectural guideline parallels what is required in real-time multi-sensor fusion.

FPGAs are well positioned to mitigate PCIe bottlenecks by performing substantial preprocessing, filtering, feature extraction, decimation, or compression in situ before forwarding data to GPUs or CPUs. In many sensor fusion workloads, only sparse features (detections, regions of interest, transformed coefficients) need to cross PCIe rather than full waveform or uncompressed image data. Surveys of FPGA-based sensor systems describe this division of labor as a primary benefit: FPGA logic reduces raw data volume by orders of magnitude before sending small fused feature sets to a GPU or CPU for high-level inference [1,2].

Additionally, some modern devices integrate FPGA fabric and CPU cores on a single chip connected by an internal network-on-chip (NoC) rather than PCIe. For example, adaptive SoCs based on Versal AI Core architectures provide AXI-based NoC links between programmable logic and embedded compute engines with deterministic latencies significantly lower than external PCIe links [7]. This reduces interconnect overhead, avoids PCIe congestion entirely for FPGA–CPU interactions, and improves determinism for front-end sensor processing.

GPUs and CPUs are highly dependent on PCIe and similar interconnects for sensor ingest and accelerator communication. These links impose bandwidth and latency constraints that are difficult to eliminate in multi-sensor, high-rate systems. FPGAs, by contrast, can avoid these bottlenecks by residing adjacent to the sensors, performing substantial preprocessing locally, and forwarding only compressed or fused outputs. In heterogeneous CPU–FPGA–GPU systems, the FPGA serves as a front-end buffer, preprocessing engine, and bandwidth-reduction stage, relieving PCIe pressure and enabling scalable multi-sensor fusion.

17. Mathematical Model

Let's formalize these dynamics. Let a sensor fusion system generate a data stream with sustained input rate R_{in} (bytes per second). When a GPU is used as the primary compute engine, the data must cross the interconnect with bandwidth B and latency L.

The time required to transfer a buffer of size D bytes across PCIe is

$$T_{xfer} = L + \frac{D}{B}$$

where
B is coded throughput of the link and
L is fixed per-transfer latency including PCIe packetization and DMA setup.
In steady state, the maximum sustainable input rate to the GPU is

$$R_{max} \approx \frac{D}{T_{xfer}} = \frac{D}{L + D/B} = \frac{B}{1 + BL/D}$$

As D becomes small, BL/D grows and the denominator increases. This yields the common observation that GPUs are inefficient for small batches, because

$$R_{max} \to \frac{D}{L} \quad \text{as} \quad D \to 0$$

which is far below the rated bandwidth B.

When multiple sensors share the same PCIe fabric, let there be N independent streams each with rate R_i. The aggregate load on the bus is

$$R_{agg} = \sum_{i=1}^{N} R_i$$

The bus is saturated when

$$R_{agg} \geq B$$

and arbitration produces a stall time

$$T_{stall} \approx \frac{R_{agg} - B}{B}, T_{window}$$

for each scheduling window T_{window} used by the PCIe switch. This stall time directly adds to end-to-end latency and produces measurable jitter.

The GPU effective compute latency for a batch of size D is therefore

$$T_{gpu} = T_{xfer} + T_{queue} + T_{kernel}$$

where T_{queue} is variable and depends on PCIe congestion and GPU scheduling.

FPGA preprocessing changes these equations. If an FPGA reduces the incoming stream by a factor ρ (for example via FFT decimation, thresholding, ROI extraction, or feature reduction), then the effective data rate sent to the GPU is

$$R_{reduced} = \rho R_{in}$$

with $0 < \rho \ll 1$ for strong front-end filtering.
The PCIe transfer time becomes

$$T_{xfer,FPGA} = L + \frac{\rho D}{B}$$

and the GPU load becomes

$$R_{reduced,max} \approx \frac{\rho B}{1 + BL/(\rho D)}$$

This shows the key architectural result:
As ρ decreases,
PCIe pressure falls proportionally,
transfer latency collapses toward L,

and GPU scheduling becomes more stable.
For CPU ingest, the model includes an additional copy step through host RAM. Let C represent host CPU copy bandwidth. The CPU-mediated transfer time is

$$T_{cpupath} = L + \frac{D}{C} + \frac{D}{B}$$

which is strictly greater than the direct FPGA–GPU RDMA path unless C and B are identical (which is not the case on any practical platform).
These models mathematically justify the observed behavior:
1. PCIe latency dominates small-batch GPU fusion pipelines.
2. Aggregate sensor bandwidth pushes GPU fusion past saturation.
3. FPGA front-end reduction improves throughput, stability, and determinism.

18. Example Use Cases and Architectures

Let's look at how these architectures play out in practice. Real-world sensor fusion architectures provide concrete examples of how CPUs, GPUs, and FPGAs are deployed together, each performing tasks suited to its architectural strengths. Across defense sensing, autonomous vehicles, robotics, and scientific data acquisition, a consistent pattern appears: FPGAs ingest and preprocess heterogeneous sensor streams; GPUs execute large-scale inference and perception; CPUs orchestrate system logic and perform higher-level data association or planning.

In defense and remote sensing systems, combining radar with EO/IR imagery is a common requirement. Radar front ends produce wideband digitized I/Q samples at high rates that must undergo digital downconversion, filtering, Doppler or range–Doppler processing, and thresholding before further interpretation. FPGA-based radar systems implement these algorithms in fully pipelined form with deterministic latency, enabling real-time detection and tracking. Surveys of FPGA-based sensor systems describe such architectures as canonical use cases, where radar pipelines and imaging pipelines coexist in programmable logic and export compact features to higher-level processors [1,2]. Similar patterns appear in remote sensing platforms where FPGA preprocessing reduces raw data volume and aligns RF and EO/IR data streams before multi-frame association or classification.

Autonomous driving and robotics systems similarly employ heterogeneous fusion. Vehicle platforms require simultaneous processing of camera, LiDAR, radar, GNSS, and inertial measurements. GPUs serve as the primary inference engines for deep neural networks performing object detection, segmentation, and scene interpretation, but cannot directly ingest or time-align raw sensor streams. FPGA-based front ends, or ASIC equivalents, commonly perform triggering, buffering, rectification, pre-filtering, and timestamping for multiple sensors. Embedded navigation systems combining GNSS, IMU, and visual sensors demonstrate this architecture clearly: FPGA or dedicated timing hardware provides sub-millisecond alignment and low-latency pre-processing, while CPUs and GPUs execute optimization, mapping, and inference [14].

A related example arises in scientific data acquisition systems, particularly in high-energy physics, where distributed detector elements must be synchronized and fused in real time. The timing system for the JUNO neutrino experiment employs an FPGA-based IEEE 1588 synchronization network that achieves nanosecond-level timestamp alignment across thousands of channels [11]. FPGAs also perform streaming feature extraction and trigger logic before transmitting compact event summaries to CPUs and GPUs for further reconstruction. Machine-learning-based trigger studies report that FPGA inference pipelines can match GPU event throughput while consuming significantly less energy, showing that for highly structured streaming workloads, FPGAs offer both deterministic timing and high efficiency [6].

These examples share a common architectural pattern:
• FPGAs implement the sensor front end, directly interfacing with radar front ends, cameras, LiDAR, inertial sensors, and high-speed ADCs. They perform deterministic preprocessing, digital filtering, decimation, and timestamping in spatially parallel pipelines.
• GPUs execute high-throughput analytics, particularly deep neural inference, dense linear algebra, and other computationally heavy but less time-critical tasks. These benefit from the GPU's massive parallelism but are tolerant of millisecond-scale latency and batching.
• CPUs manage system-level logic and fusion decision-making, performing multi-sensor association, hypothesis tracking, path planning, and communication with higher-level control functions.

This division of labor aligns with the known characteristics of each architecture: FPGAs excel at deterministic, low-latency, multi-

stream ingest; GPUs excel at high-throughput compute once data has been preprocessed and structured; and CPUs excel at complex control and orchestrating heterogeneous accelerators. Published FPGA-based sensor system surveys, multi-sensor navigation studies, and data-acquisition timing networks all report variants of this same triangular architecture [1,2,14,11].

As sensor bandwidths and sensor counts continue to rise, these architectures increasingly depend on FPGA front ends for data-rate reduction, timestamp precision, and real-time filtering. Correspondingly, GPU subsystems continue to scale for higher inference throughput, and CPU subsystems integrate with the FPGA and GPU elements to provide global fusion logic. The resulting heterogeneous architecture is now the predominant pattern for modern real-time multi-sensor fusion systems.

19. Comparative Performance Summary

The following table summarizes the key performance characteristics of CPUs, GPUs, and FPGAs for sensor fusion applications, based on the preceding discussion and cited studies.

Characteristic	CPU (General-Purpose Processor)	GPU (Graphics Processor)	FPGA (Field-Programmable Gate Array)
Throughput (operations/sec)	Limited by a small number of cores; typically ~10^9–10^{10} effective ops/sec for DSP and vision workloads.	Very high peak throughput (~10^{13}–10^{14} ops/sec) when thread/batch parallelism is large. Throughput drops under irregular or I/O-bound workloads.	High streaming throughput; one-sample-per-clock pipelines achieve ~10^{12}–10^{13} ops/sec. Efficient for structured DSP and fusion pipelines.
Latency & determinism	Tens of microseconds to milliseconds; substantial jitter from OS scheduling, cache misses, and interrupts.	Millisecond-scale latency; kernel launches and batching introduce variability. Cannot guarantee uniform deadlines.	Deterministic, cycle-accurate pipelines; microsecond-scale end-to-end latency with negligible jitter. Suitable for hard real-time fusion.
Parallelism & concurrency	Limited SIMD and thread-level parallelism; many sensor streams contend for shared execution resources.	Massive data-parallel throughput for homogeneous kernels. Multiple sensor streams time-share hardware, introducing interference and scheduling overhead.	True spatial parallelism: separate hardware pipelines per sensor. All pipelines operate concurrently with no context switching.
Power efficiency (ops/W)	Lowest energy efficiency for numerical workloads; general-purpose hardware wastes power on unused structures.	Moderate to low efficiency under load (250–400+ W typical). Efficiency falls for small batches and streaming workloads with limited parallelism.	Highest measured efficiency: 5–10× CPU energy efficiency; 2–10× GPU energy efficiency for structured DSP/vision workloads (published data).
Multi-sensor handling	Threads and interrupts must service each sensor; scalability degrades with high-rate asynchronous inputs.	Cannot directly ingest raw sensors; requires CPU/FPGA front end. High-rate multi-sensor I/O overwhelms GPU and its memory subsystem.	Ideal for multi-sensor ingest: hardware timestamping, format conversion, buffering, and sync performed in parallel across many sensors.
Interconnect / PCIe impact	CPU can saturate quickly when feeding GPUs or moving data to accelerators; PCIe and interrupts introduce jitter.	Strong PCIe dependency; PCIe bandwidth/latency often limits fusion throughput. GPUDirect RDMA helps but does not eliminate PCIe bottlenecks.	Can reside adjacent to sensors and avoid PCIe entirely. As accelerator, performs in-situ reduction to minimize PCIe traffic.
Role in fusion architectures	High-level system control, data association, and orchestration.	High-throughput perception and inference (DNNs, filtering, dense math).	Front-end ingest, preprocessing, timestamping, and real-time fusion of heterogeneous sensors.

20. Discussion

The comparative data show that FPGAs provide strong advantages for sensor fusion workloads where latency, determinism, multi-stream concurrency, and throughput-per-watt dominate. Published filtering and vision-kernel comparisons report FPGA speedups of 20× relative to CPUs and 1.2×–2× relative to GPUs on structured DSP kernels, with 2–10× higher energy efficiency on average. High-energy physics trigger pipelines demonstrate FPGA parity with GPU event throughput but with significantly lower energy per event. Large-scale server deployments show that FPGAs can achieve near order-of-magnitude improvements in performance per watt over CPUs for streaming workloads.

CPUs remain indispensable for high-level orchestration, but cannot service many high-rate sensors without eventually saturating their interrupt and memory subsystems. GPUs provide unmatched bulk throughput, but require batching, incur PCIe latency, and do not natively handle raw sensor ingest. FPGAs, by contrast, operate on continuous streams with deterministic timing, attach directly to heterogeneous sensor interfaces, and reduce data volume before sending compact feature sets to GPUs or CPUs. These factors explain why modern real-time fusion architectures consistently adopt FPGA front ends coupled with GPU back ends and CPU supervisory control.

21. End-to-End Fusion Timing Equation

For a general heterogeneous fusion system, the end-to-end latency from physical sensor sampling to fused output can be written as the sum of four components:

$$L_{total} = L_{sens} + L_{front} + L_{xfer} + L_{back}$$

Here L_{sens} is the sensor-internal latency between physical measurement and digital availability, L_{front} is the latency of the front-end processing stage that first receives digital samples, L_{xfer} is the interconnect and buffering latency between front end and back end, and L_{back} is the latency of the back-end fusion and inference stage.

In a CPU-only pipeline, the sensor connects directly to the CPU. In that case

$$L_{front,CPU} = L_{ingest} + L_{queue} + L_{sched}$$

$$L_{back,CPU} = L_{proc}$$

and the total latency becomes

$$L_{total,CPU} = L_{sens} + L_{ingest} + L_{queue} + L_{sched} + L_{proc}$$

Using the earlier CPU model with service time S_{CPU} and utilization ρ_{CPU}, an effective approximation is

$$L_{total,CPU} \approx L_{sens} + L_{ingest} + L_{sched} + \frac{S_{CPU}}{1 - \rho_{CPU}}$$

For a CPU plus GPU architecture without FPGA preprocessing, the sensor connects to the host, and the GPU acts as the main fusion engine. The front end is CPU ingest and buffering, and the back end is GPU compute. The end-to-end latency becomes

$$L_{total,CPU+GPU} = L_{sens} + L_{ingest} + L_{queue} + L_{sched} + L_{xfer,GPU} + L_{GPU}$$

Using the earlier expressions for PCIe and GPU behavior, the transfer and GPU components can be written as

$$L_{xfer,GPU} = L + \frac{D_{in}}{B} + \frac{D_{out}}{B}$$

$$L_{GPU} = L_{batch} + L_{launch} + \frac{T_{kernel}(B)}{B}$$

so that

$$L_{total,CPU+GPU} \approx L_{sens} + L_{ingest} + L_{queue} + L_{sched} + L + \frac{D_{in}}{B} + \frac{D_{out}}{B} + L_{batch} + L_{launch} + \frac{T_{kernel}(B)}{B}$$

This expression makes explicit the dependence of end-to-end latency on PCIe terms, batching, kernel launch, and GPU occupancy.

For a FPGA plus GPU plus CPU architecture, the FPGA is the deterministic front end, the GPU is the high-throughput back end, and the CPU provides orchestration, but is not on the critical data path for each sample. The FPGA implements a streaming pipeline with depth D and clock period T_{clk}, and reduces the data volume by a factor ρ before sending it to the GPU. The front-end latency is then

$$L_{front,FPGA} = L_{iface} + D, T_{clk}$$

The reduced transfer and GPU terms are

$$L_{xfer,FPGA \to GPU} = L + \frac{\rho D_{in}}{B} + \frac{D_{out}}{B}$$

$$L_{GPU,stream} = L_{batch,eff} + L_{launch,eff} + \frac{T_{kernel,eff}(\rho B)}{\rho B}$$

where $L_{batch,eff}$ and $L_{launch,eff}$ represent effective batching and launch delays after FPGA reduction, and $T_{kernel,eff}$ is the kernel execution time on the reduced workload. The end-to-end latency becomes

$$L_{total,FPGA+GPU} = L_{sens} + L_{iface} + D, T_{clk} + L + \frac{\rho D_{in}}{B} + \frac{D_{out}}{B} + L_{batch,eff} + L_{launch,eff} + \frac{T_{kernel,eff}(\rho B)}{\rho B}$$

In the limiting case of a fully FPGA-resident fusion pipeline, where the FPGA performs both front-end processing and the core fusion logic and only compact fused results are returned to a CPU, the GPU terms vanish and the PCIe term is either absent or negligible. The end-to-end latency reduces to

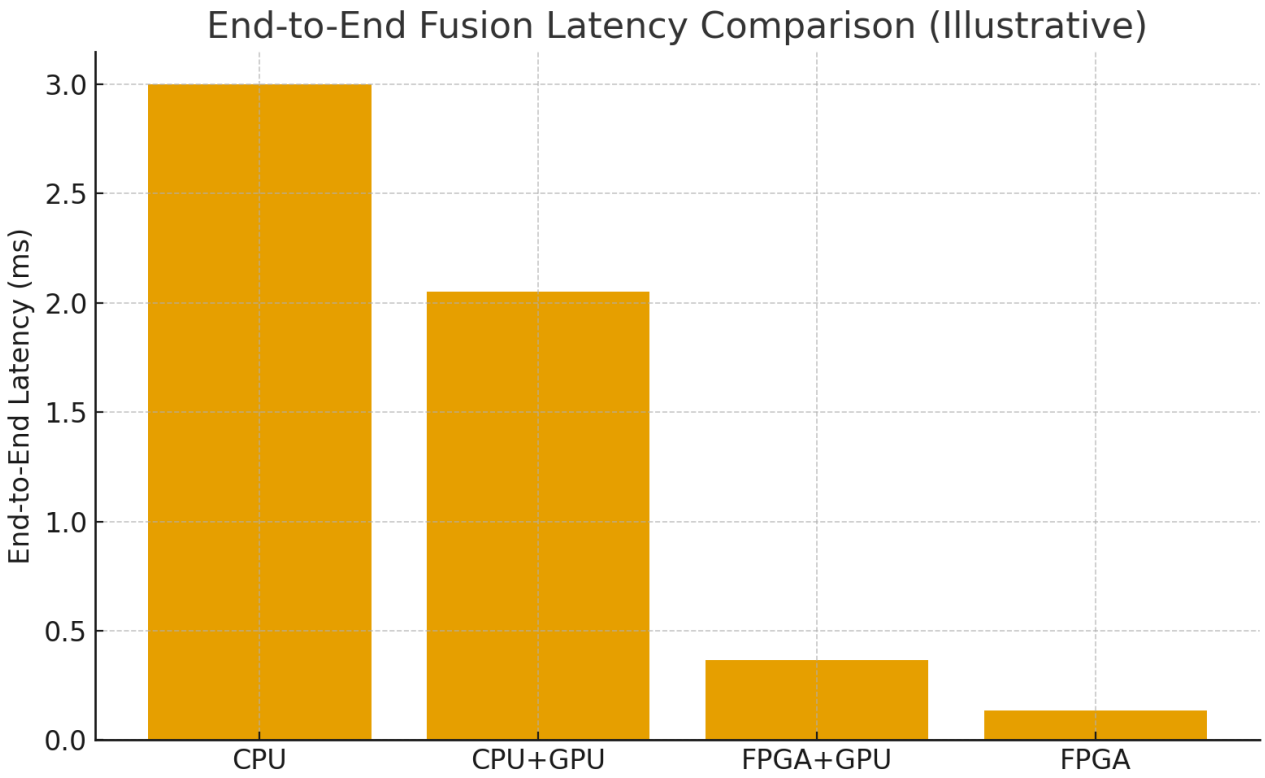
$$L_{total,FPGA} = L_{sens} + L_{iface} + D, T_{clk} + L_{out,host}$$

where $L_{out,host}$ is a small fixed reporting latency to the host processor, often much smaller than $D,\ T_{clk}$.

These expressions formalize the observed hierarchy

$$L_{total,FPGA} \ll L_{total,FPGA+GPU} \ll L_{total,CPU+GPU} \lesssim L_{total,CPU}$$

for real-time fusion workloads where $D,\ T_{clk}$ is on the order of microseconds, where PCIe and batching terms dominate GPU execution, and where CPU scheduling and queuing effects dominate software-only designs.



22. FPGA Sensor Performance Advantage

The preceding sections establish that for real-time sensor fusion workloads involving high data rates, strict latency requirements, multi-stream concurrency, and energy constraints, FPGAs consistently demonstrate performance advantages over CPUs and often over GPUs. These advantages arise from architectural differences grounded in spatial parallelism, pipeline determinism, and proximity to the sensor interfaces.

In throughput, empirical studies show that FPGAs can match or exceed GPU performance for structured streaming workloads. FIR filter benchmarks, a close proxy for radar, EO/IR pre-processing, and similar DSP kernels, demonstrated that FPGA implementations complete the same workload at twice the speed as optimized GPU versions (approximately 0.004 s versus 0.008 s), while the CPU required 0.107 s for the identical operation [5]. Other vision-kernel comparisons report FPGA speedups of 1.2× to more than 20× over CPUs and GPUs depending on the kernel's structure and available parallel replication [4]. For workloads characterized by fixed pipelines, continuous sampling, and minimal branching, typical of raw sensor ingest and low-level fusion, FPGAs offer competitive or superior throughput compared to GPUs while vastly outperforming CPUs.

Low latency and low jitter are critical in fusion scenarios such as radar–EO/IR alignment, autonomous navigation, target tracking, and guidance. FPGA pipelines operate in deterministic clocked datapaths with end-to-end delays of microseconds, independent of system load. In contrast, GPUs incur kernel launch latency, batching delay, and PCIe transfer overhead, resulting in millisecond-scale or larger variability. CPUs experience even greater jitter due to operating-system scheduling and cache behavior. Studies of FPGA-based timing networks demonstrate that FPGA logic can maintain nanosecond-level synchronization and deterministic processing even under heavy multi-stream load [11-13]. These properties make FPGAs well suited for hard real-time fusion loops where consistent response time is required every cycle.

Energy efficiency further differentiates the architectures. Comparative measurements show FPGAs achieving two to ten times higher throughput per watt than GPUs for structured DSP and vision workloads, and significantly higher still relative to CPUs [4]. In high-energy physics trigger applications, which are computationally intensive, high-bandwidth, real-time pipelines, FPGAs delivered event rates comparable to GPUs while consuming far less energy per processed event, yielding better performance-per-watt efficiency [6]. Datacenter-scale FPGA deployments report up to an order-of-magnitude improvement in throughput per watt relative to CPU-only configurations for streaming workloads [3]. For deployed systems where power is constrained, such as UAS, ISR payloads, and edge

autonomous systems, this efficiency becomes a decisive factor.

Multi-sensor scalability also favors FPGAs. Sensor fusion systems routinely integrate high-rate cameras, LiDAR, radar, GNSS, and IMUs, each with distinct timing and bandwidth requirements. FPGA devices can instantiate multiple independent pipelines, one per sensor, and operate on them concurrently. Published sensor-system surveys consistently emphasize that FPGA-based ingestion removes CPU bottlenecks, provides precise timestamping, and enables per-stream buffering and filtering with deterministic behavior even when sensor count increases [1,2]. GPUs cannot directly ingest raw sensor data and rely on a CPU or FPGA to pre-align, batch, or transfer data. CPUs must perform this ingest in software and degrade rapidly as interrupt and buffer pressure increases.

Finally, interconnect constraints magnify these differences. GPU-based fusion pipelines are strongly dependent on PCIe bandwidth and latency. Even with GPUDirect RDMA, PCIe remains the limiting factor in many multi-sensor environments [9]. FPGAs mitigate this by performing local preprocessing and data reduction (FFT, channelization, ROI extraction, decimation) before transmitting compact fused features rather than raw sensor data. In heterogeneous architectures, this lowers PCIe load, improves determinism, and scales more effectively with sensor-channel count.

The available evidence demonstrates that FPGAs have clear performance advantages over CPUs and significant advantages over GPUs in sensor fusion scenarios characterized by continuous high-rate data streams, strict or hard real-time latency requirements, large numbers of asynchronous sensors, limited power budgets, the need for precise timestamping and deterministic behavior, and heavy front-end DSP, filtering, or pre-alignment.

GPUs remain unmatched for bulk parallel compute, especially deep neural inference and dense linear algebra. CPUs remain essential for control, decision-making, and managing heterogeneous resources. However, for the ingestion, reduction, alignment, and early fusion of sensor data, FPGAs consistently provide the most favorable combination of throughput, determinism, scalability, and energy efficiency.

23. Future Work

Future work is expected to include various implementations of parallel ISR pipelines, to measure both performance and fusion precision across CPU, GPU, and FPGA implementations under identical sensor workloads. The objective is to move from qualitative architectural arguments to quantitative efficacy measurements that directly characterize throughput, latency, energy efficiency, time alignment, and fused-state accuracy for representative RF, EO/IR, and LiDAR fusion tasks.

First, I present no original benchmark data. The performance comparisons draw on published studies that used different hardware platforms, different workloads, and different measurement methodologies. The numeric comparison table in the throughput-per-watt section uses illustrative figures derived from reported ranges, not measured values from a single controlled experiment. Until identical fusion algorithms are benchmarked on identical input data across all three architectures using instrumented measurement, the quantitative advantage factors cited here should be understood as indicative rather than definitive.

Second, I have not modeled development complexity or time-to-deployment for FPGA designs, and that matters. FPGA development cycles are substantially longer than equivalent software implementations on CPUs or GPUs. RTL design, simulation, synthesis, place-and-route, timing closure, and hardware verification require specialized skills and tools. High-level synthesis (HLS) has reduced this gap but has not eliminated it. For organizations evaluating hardware platforms, development cost and schedule risk may outweigh raw performance advantages in some contexts.

Third, the mathematical models in this paper assume ideal conditions. The FPGA latency model assumes no backpressure, no resource contention, and no dynamic reconfiguration overhead. The GPU model assumes a simple batch-and-launch paradigm, but modern GPU runtimes (e.g., CUDA Graphs, persistent kernels) can substantially reduce launch overhead. The CPU model does not account for real-time operating systems, kernel bypass techniques, or DPDK-style polling modes that reduce scheduling jitter. Future work should validate these models against measured behavior on contemporary hardware.

Fourth, I have not addressed emerging processing architectures that may alter the competitive landscape. Neural processing units (NPUs), tensor processing units (TPUs), and neuromorphic processors represent alternative approaches to low-power, high-throughput inference. Some of these architectures may prove competitive with FPGAs for specific fusion subtasks, particularly neural-network-based perception, and future comparisons should include them.

Fifth, I have not quantified the impact of FPGA partial reconfiguration overhead, which is relevant in systems that must dynamically switch between different sensor processing modes. Reconfiguration latency, even in modern devices with partial reconfiguration support, can range from milliseconds to tens of milliseconds, which may interrupt real-time processing.

Finally, while I argue for heterogeneous architectures, the optimal partitioning of workloads between FPGA, GPU, and CPU for a given fusion problem remains an open design question. No general-purpose methodology exists for determining which stages of a fusion pipeline should be implemented in which architecture, and this partitioning decision significantly affects overall system performance, cost, and maintainability.

In this author's experience, even in the absence of such controlled studies, the operational differences between these architectures are rarely subtle. As channel load increases and sensor throughput approaches saturation, there are distinct tipping points at which fusion performance degrades rapidly and, in real-time settings, can fail outright. These effects are observable to operators long before formal benchmarks are performed. This does not diminish the need for rigorous measurement, but it underscores that the fundamental architectural distinctions outlined here tend to manifest clearly in practice, particularly under high-rate, multi-sensor conditions.

24. Conclusion

I have argued in this paper, with quantitative support from the published literature, that FPGAs represent the most effective front-end architecture for real-time multi-sensor fusion. The case rests on four mutually reinforcing advantages: deterministic microsecond-scale latency that CPUs and GPUs cannot match; spatial parallelism that scales naturally with sensor count; energy efficiency that is two to ten times better than GPUs and far better than CPUs for structured streaming workloads; and an ability to perform substantial in-situ data reduction that relieves PCIe and system bandwidth pressure before data ever reaches a back-end compute element.

I want to be precise about what this recommendation does not say. GPUs are irreplaceable for large-scale neural inference, dense linear algebra, and any workload that genuinely benefits from thousands of identical threads operating in lockstep. CPUs are irreplaceable for orchestration, decision logic, and tasks requiring flexible general-purpose computation. The recommendation is not to replace either, but to place FPGAs where the data is born: at the sensor interface, in the front-end pipeline, and in the time-alignment and reduction stages that condition data for everything downstream.

The heterogeneous architecture that results, with FPGA front end, GPU inference back end, and CPU supervisory control, is already the predominant pattern in serious multi-sensor fusion deployments, from defense ISR payloads to autonomous vehicle platforms to high-energy physics trigger systems. Emelita C. Baylon the mathematical foundations of that architecture and provides the empirical benchmarks that justify it. My expectation is that as sensor bandwidths continue to grow and latency requirements tighten, the architectural arguments made here will only strengthen.

References

1. De La Piedra, A., Braeken, A., & Touhafi, A. (2012). Sensor systems based on FPGAs and their applications: A survey. *Sensors, 12*(9), 12235-12264.
2. García, G. J., Jara, C. A., Pomares, J., Alabdo, A., Poggi, L. M., & Torres, F. (2014). A survey on FPGA-based sensor systems: towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing. *Sensors, 14*(4), 6247-6278.
3. Putnam, A., Caulfield, A. M., Chung, E. S., Chiou, D., Constantinides, K., Demme, J., ... & Burger, D. (2014). A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News, 42*(3), 13-24.
4. Qasaimeh, M., Denolf, K., Lo, J., Vissers, K., Zambreno, J., & Jones, P. H. (2019, June). Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels. In *2019 IEEE international conference on embedded software and systems (ICESS)* (pp. 1-8). IEEE.
5. Arucu, M., & Iliev, T. (2025). Performance Evaluation of FPGA, GPU, and CPU in FIR Filter Implementation for Semiconductor-Based Systems. *Journal of Low Power Electronics and Applications, 15*(3), 40.
6. Giasemis, F. I., Lončar, V., Granado, B., & Gligorov, V. V. (2025, June). Comparative analysis of FPGA and GPU performance for machine learning-based track reconstruction at LHCb. In *2025 23rd IEEE Interregional NEWCAS Conference (NEWCAS)* (pp. 533-537). IEEE.
7. AMD. (2020). *Versal AI Core Series Product Selection Guide*. AMD Xilinx.
8. NVIDIA, N. (2020). *a100 tensor core gpu datasheet.*
9. NVIDIA. (2021). "GPU Performance Background User's Guide." Deep Learning Performance Documentation. Accessed November

26, 2025.
10. Sander, B. (2025). “Understanding Peak, Max-Achievable and Delivered FLOPs.” ROCm Performance Blog. Accessed November 26, 2025.
11. Pedretti, D., Bellato, M., Isocrate, R., Bergnoli, A., Brugnera, R., Corti, D., ... & Verde, G. (2019). Nanoseconds timing system based on IEEE 1588 FPGA implementation. *IEEE Transactions on Nuclear Science, 66*(7), 1151-1158.
12. Nagra, A. S., Allahi, I., Pasha, M. A., & Masud, S. (2022). Design and FPGA based implementation of IEEE 1588 precision time protocol for synchronisation in distributed IoT applications. *Australian journal of electrical and electronics engineering, 19*(2), 1-9.
13. Li, F., Liu, W., Qi, Y., Li, Q., & Liu, G. (2023). An enhanced method for nanosecond time synchronization in IEEE 1588 precision time protocol. *Processes, 11*(5), 1328.
14. Liu, C., Xiong, S., Geng, Y., Cheng, S., Hu, F., Shao, B., ... & Zhang, J. (2023). An embedded high-precision gnss-visual-inertial multi-sensor fusion suite. *NAVIGATION: Journal of the Institute of Navigation, 70*(4).
15. Patil, M., Ng, S. F., & Udkar, S. (2023). Comparative study of FPGA and GPU for high-performance computing and AI. *ESP International Journal of Advancements in Computational Technology (ESP-IJACT), 1*(1), 37-46.