# **FPGA Acceleration in Finance: Low-Latency Algorithmic Trading and Technical Analysis**

# Field-Programmable Gate Arrays (FPGAs) are increasingly deployed in financial markets to accelerate algorithmic trading. The competitive landscape of **high-frequency trading (HFT)** and real-time market analysis demands the lowest possible latency and high throughput. Unlike general-purpose CPUs or even GPUs, FPGAs can be configured as custom hardware tailored to specific trading algorithms, including technical analysis indicators like RSI, MACD, and moving averages, to generate trading signals with minimal delay. In this report, we provide a comprehensive overview of FPGA use in finance – from low-latency tick data processing to technical indicator acceleration – and evaluate how a university-level FPGA project implementing technical analysis (e.g. RSI and EMA on Verilog) aligns with industry standards. We review the state-of-the-art in both commercial solutions and academic research, compare hardware acceleration against CPU/GPU performance, and recommend steps to evolve the project into a publishable paper and a viable prototype. The analysis is structured to highlight reconfigurable computing benefits for trading, reflecting the interests in hardware accelerators for finance.

## **FPGA Use in Financial Markets: Low Latency and Real-Time Processing**

Modern financial trading is extremely time-sensitive – firms compete to react to market data in **nanoseconds to microseconds**. FPGAs have emerged as a critical tool in achieving *ultra-low latency* for several key tasks in electronic trading:

* **Market Data Feed Handling:** Exchanges send out *tick-by-tick* prices and order book updates at very high rates. FPGA-based feed handlers can parse and aggregate these feeds faster than software. For example, NovaSparks’ FPGA-based ticker plant *normalizes direct exchange feeds with full order book building in under one microsecond*, even under peak load. This level of speed, sustained regardless of bursty data, is hard to achieve with CPU-based feed handlers. By processing network packets and decoding market protocols entirely in hardware, FPGAs eliminate software overhead and achieve wire-speed data handling.
* **Low-Latency Algorithmic Trading:** In **high-frequency trading**, where strategies aim to capitalize on minuscule price differentials, *every nanosecond of advantage counts*. FPGAs can execute trading logic in hardware, bypassing the indeterministic delays of operating systems. Industry platforms report end-to-end FPGA trading systems that handle *market data, strategy logic, and order execution in a few tens of nanoseconds* total. Notably, a recent AMD Alveo accelerator card achieved a record **13.9 ns tick-to-trade latency**, nearly halving the previous best of ~24 ns. This “tick-to-trade” interval measures from the arrival of the last data needed to decide, to the issuance of an order – a critical metric in HFT. Such performance approaches the realm of custom ASICs, giving FPGAs an ASIC-like edge while retaining reconfigurability.
* **Real-Time Signal Generation:** Beyond raw feed handling and order execution, FPGAs enable on-the-fly computation of *trading signals* or indicators as data streams in. Technical analysis indicators (e.g. moving averages, oscillators) traditionally computed in software can be offloaded to hardware for instantaneous calculation. This is valuable not only in HFT but also in **algorithmic trading strategies** that scan live data for signals. For instance, an FPGA can maintain and update indicators like RSI or MACD tick-by-tick so that a buy/sell decision can be triggered the moment conditions are met, without waiting for software to poll and compute. Academic work has shown that moving a technical indicator from software to FPGA can yield massive speedups – one study reported a hardware MACD implementation achieving **30× faster computation than a software version** on the same data. By streaming price data through parallel arithmetic pipelines (for sums, differences, etc.), an FPGA can output indicator values with only a few clock cycles of latency, essentially in real-time relative to market data.
* **Deterministic Performance:** Importantly, FPGAs bring a level of *determinism and consistency* that is crucial under volatile market conditions. Hardware logic in an FPGA will always follow the same fixed data path and timing for a given task. Even during sudden bursts of activity when networks and CPUs are overloaded, an FPGA’s performance remains steady – it *always passes through the same sequence of states, yielding predictable latency* for each operation. This contrasts with CPU-based trading software whose latency can spike unpredictably due to context switches, interrupts, or cache misses during busy periods. The deterministic nature of FPGAs ensures trading signals and orders are generated without random delays, instilling confidence in time-critical strategies.

In summary, FPGAs in finance serve as **hardware accelerators** for low-latency, high-throughput tasks: ingesting market feeds, computing analytics, and executing orders faster than conventional systems. Their *parallel architecture* allows multiple data processing tasks to run concurrently without contention, a crucial advantage when thousands of instruments’ data must be tracked simultaneously. As a result, a single FPGA can maintain numerous indicator calculations or order book updates in parallel, where a CPU might struggle to sequentially iterate through them in time. These attributes have made FPGAs a mainstay in cutting-edge trading infrastructure. Next, we examine the ecosystem of solutions – from leading vendors to open-source projects and academic contributions – that leverage FPGAs for financial trading.

## **Industry Adoption: Commercial Solutions and Open Initiatives**

The use of FPGAs in trading has matured, with several **commercial vendors, open-source frameworks, and academic groups** advancing the state of the art. This section surveys prominent examples:

### **Commercial FPGA Trading Solutions**

A number of specialized firms provide FPGA-based trading hardware or intellectual property, catering to banks, hedge funds, and exchanges that require speed:

* **NovaSparks (NovaTick):** *NovaSparks is a recognized leader in FPGA-based ultra-low latency trading solutions*. Their flagship NovaTick appliance employs FPGAs to handle market data feeds for dozens of exchanges. NovaTick’s fully FPGA ticker plant can ingest and normalize feeds (including building entire order books) in **under 1 µs** per tick, even during peak traffic. Moreover, NovaSparks offers a *“pure FPGA Tick-to-Trade”* solution, extending hardware acceleration through the entire trading cycle – from feed handling to decision logic to order execution. This demonstrates the viability of all-in-hardware trading engines meeting extreme speed demands.
* **Enyx (nxAccess):** Enyx provides FPGA IP cores and cards for finance, such as the **nxAccess** platform. This embedded FPGA solution includes a feed handler with order book construction and allows users to deploy their custom trading algorithms on the same FPGA. By having the *full depth-of-market view updated in hardware*, algorithms can make decisions on the most up-to-date state of the market without software latency. Enyx’s products illustrate how vendors deliver pre-built FPGA components (feed parsers, book builders, DMA engines) that adhere to industry protocols, letting firms focus on their proprietary trading logic.
* **Algo-Logic Systems:** A niche player focusing on FPGA **tick-to-trade systems**, Algo-Logic offers pre-engineered cores for specific exchange feeds and order execution. For example, they have an FPGA-based CME (Chicago Mercantile Exchange) feed handler that processes the CME’s Market Data Protocol 3.0 with sub-microsecond latency. Their latest generation claims a *sub-500 nanosecond end-to-end tick-to-trade latency* by combining feed parsing, decision logic, and a “turbo” order execution on FPGA. These numbers underscore how far hardware acceleration can push performance beyond what even the fastest software can do.
* **Network Appliance Vendors (Arista, Cisco):** FPGA technology has also been integrated into network devices for trading. Arista Networks (after acquiring low-latency specialists like Metamako) offers switches and network interface cards with embedded FPGAs that can perform cut-through forwarding and filtering in nanoseconds. Such devices can be programmed to, for example, filter only relevant stock symbols or even embed simple trading logic directly in the network path. This *pre-processing at the network level* using FPGAs further reduces latency by offloading work before data even reaches the server’s CPU.
* **Major FPGA Manufacturers:** Both Xilinx (now part of AMD) and Intel (via its Altera FPGA line) actively promote their hardware for finance. AMD’s **Alveo** accelerator cards (e.g. the Alveo UL3524 FinTech card) are purpose-built for electronic trading, featuring ultra-low latency transceivers and co-located FPGA+networking on a PCIe card. These cards advertise single-digit nanosecond latency capabilities and come with reference designs to implement trading systems. Similarly, Intel’s high-end Stratix 10 FPGAs are deployed in trading systems, sometimes paired with specialized network IP for ultra-fast Ethernet and PCIe connectivity. The FPGA manufacturers also provide development toolchains and reference IP (such as low-latency 10GbE MACs, FIX protocol engines, etc.) to help financial customers build solutions more easily.

In the commercial realm, **speed and determinism are the key selling points**. Vendors highlight how their FPGA solutions can outperform software by orders of magnitude. For instance, Magmio (a trading platform provider) notes that FPGAs *“process market data and execute trades up to 1000× faster than traditional software solutions”*, operating in *“nanoseconds, not microseconds”*. This claim reflects the parallelism and minimal overhead of hardware execution. Moreover, integrated solutions often combine multiple components in FPGA: feed handling, order book management, risk checks, strategy logic, and order output. By keeping the entire pipeline in hardware, unnecessary data copies and context switches are eliminated. **Table 1** summarizes example capabilities of commercial FPGA trading solutions:

| **Vendor / Product** | **Functionality** | **Latency Performance** |
| --- | --- | --- |
| *NovaSparks NovaTick* | FPGA ticker plant (feed handler + book) | < 1 µs feed-to-book update |
| *AMD Alveo UL3524* | FPGA accelerator card for trading | ~14 ns tick-to-trade (STAC-T0) |
| *Algo-Logic T2T System* | Full tick-to-trade engine (CME feed) | ~500 ns tick-to-trade |
| *Enyx nxAccess* | FPGA feed handler + user algo sandbox | ~1 µs feed handling; user logic runtime (ns–µs) |
| *Xilinx AAT Reference* | Open reference design (feed + order book + algo) | ~1–2 µs (baseline demo, can be tuned) |

*(Latency figures are approximate and scenario-dependent. 1 µs = 1000 ns.)*

### **Open-Source and Community Efforts**

In parallel with commercial offerings, there have been **open-source initiatives** and community projects that apply FPGAs to trading problems:

* **Xilinx Accelerated Algorithmic Trading (AAT) Reference Design:** Xilinx released an open-source HLS (High-Level Synthesis) reference platform for algorithmic trading. It provides example FPGA code (in C/C++ for HLS) implementing a feed handler (for a simplified FIX feed), an order book, a pricing engine, and an order execution module – essentially a template for a basic trading system in hardware. The AAT design runs on Xilinx Alveo cards and is meant to reduce the barrier to entry for FPGA trading development. By offering this complete framework, Xilinx enables researchers or smaller firms to experiment with hardware trading without needing a large team of FPGA experts. The design is modular, so users can **plug in their own trading algorithms** (such as custom signal generators or strategies) into the provided hardware infrastructure.
* **Academic Open Projects:** Often, academic researchers publish or open-source their FPGA implementations. For example, the GitHub repository *“FPGA-Based Low-Latency Accelerator for Stock Market Indicators”* by Azaz Hassan Khan et al. provides VHDL/Verilog code for computing technical indicators (MACD, RSI, Aroon) on a Xilinx Zynq FPGA. This code corresponds to their academic paper demonstrating 30× speedups for these indicators and explores hybrid hardware/software co-design on SoC devices. Such projects serve as valuable references for how to implement financial algorithms in FPGA logic, and being open-source, they allow others (like students) to learn from and build upon them. Similarly, our foundation project – implementing RSI and moving average computations in Verilog – is part of this growing body of open-source educational work focused on *reconfigurable computing for finance*.
* **Community Knowledge Sharing:** Beyond code releases, the FPGA trading community shares knowledge through blogs and forums. For instance, **algorithmic trading communities** on Reddit or specialized blogs (e.g., the *Velvetech* blog on FPGAs in HFT) discuss practical aspects of using FPGAs, latency tuning, and compare CPU/GPU/FPGA approaches. Some FPGA engineers at trading firms also give talks or write about their experiences (e.g., Williston Hayes of Optiver presented on FPGA usage at FPL 2020). While not exactly “open-source projects,” these resources contribute to an open exchange of ideas that helps align academic and industry efforts.

Open initiatives are important for **lowering entry barriers**. They provide templates and proofs-of-concept that students and researchers can emulate rather than starting from scratch. As an example, the Xilinx AAT platform, being open and *“designed for software engineers”* with HLS, bridges the gap between a concept and a deployable FPGA trading system. It encapsulates industry best-practices (like using double data rate memory for order books, PCIe host interfaces, etc.) in an accessible form. Likewise, academic open-source indicator projects demonstrate how common trading computations can be optimized in hardware. Overall, the synergy of commercial and open efforts is pushing FPGAs in finance toward more standardized, reusable solutions.

### **Academic Research Contributions**

Despite the proprietary nature of many trading technologies, academia has made distinct contributions in this domain, often focusing on novel architectures or comparative analyses:

* **Hardware-Accelerated Technical Analysis:** A 2024 study by Ali *et al.* is one of the first to specifically target technical analysis indicators on FPGA, as noted earlier. They designed custom hardware for MACD, RSI, and Aroon indicators on a Xilinx Zynq SoC, and reported latency for processing one day of cryptocurrency tick data. Their results – MACD yielding the best performance with **30× speedup** over software – provide concrete evidence of FPGA benefits in financial signal processing. Interestingly, they also proposed *combining indicators in hardware* (e.g., running MACD and RSI in parallel on the FPGA) to exploit parallelism and generate composite signals. This highlights how FPGAs can handle multiple analytics concurrently, a theme not explored in traditional software studies of indicators. The paper also emphasizes the scarcity of prior hardware implementations in literature, underlining how this is a relatively nascent research area.
* **FPGA-Based Trading Engines and Co-design:** Several works have looked at building entire trading strategies or components in FPGA. For example, Gupta *et al.* (2024) presented an FPGA-based HFT system that achieved processing of *150,000 orders per second with considerably lower power consumption than CPU-based systems*. Their work, focused on reducing latency in financial systems, underscores not just speed but the **energy efficiency** gains (an increasingly important metric) of FPGA acceleration. Other researchers have explored hardware/software co-design: using an FPGA for the latency-critical path and a CPU for complex logic. The Zynq SoC (which includes ARM processors with an FPGA fabric) is a popular platform for such studies, as it allows partitioning trading tasks between hardware and software for optimal performance.
* **High-Level Synthesis and Algorithmic Complexity:** Academic projects also often investigate development methodologies. The use of high-level synthesis to express trading algorithms (in C/C++ or OpenCL) and then compile to FPGA has been studied as a way to shorten development time and involve non-hardware engineers. Additionally, there is interest in determining how *complex* an algorithm can be while still fitting in the tight latency budgets of trading. Klaisoongnoen *et al.* (2021) discuss that historically only relatively *simplistic transformations* could be done in real-time due to hardware limitations, but with newer FPGAs and improved tooling, more advanced computations (potentially machine learning models or more sophisticated analytics) might become feasible within the nanosecond-to-microsecond window. Such research guides the future direction – for instance, could deep learning inference for trade prediction run on FPGA within microseconds? Or can backtesting be accelerated with reconfigurable hardware? These remain active areas at the intersection of finance and reconfigurable computing.

In summary, academic contributions have validated the advantages of FPGA acceleration for trading (quantifying speedups and efficiency gains) and are expanding the horizon of what’s possible in hardware. By prototyping and measuring real-case scenarios (like computing popular indicators on actual market data), these studies provide a scientific basis that can influence industry adoption. They also serve as **education and training** grounds – many undergrad/graduate projects (like the one we built) draws inspiration from such research to implement and evaluate trading algorithms on FPGA platforms.

## **Performance Benchmarks: FPGA vs CPU/GPU in Financial Computing**

Hardware acceleration in finance is ultimately justified by performance metrics. Here we compare **latency, throughput, and power efficiency** of FPGAs against traditional processors for financial signal processing tasks:

* **Latency:** FPGAs excel at minimizing processing latency. By running at the hardware’s native clock and eliminating software overhead, they can respond to events orders of magnitude faster than CPUs. For example, an FPGA implementation of a trading strategy might have a reaction latency measured in tens of nanoseconds, whereas a highly optimized CPU-based strategy (even with kernel bypass networking) might have latency on the order of several microseconds (i.e. 1,000s of nanoseconds). Real-world benchmarks illustrate this difference: FPGA solutions have achieved ~14 ns tick-to-trade latencies in live settings, while pure software is challenged to get below ~1–5 µs (1000–5000 ns) even with specialization. In one use case, migrating a MACD indicator from software to FPGA not only sped it up 30×, but also made the processing **deterministic** – every tick incurs the same fixed delay, instead of variable delays. GPUs, on the other hand, are typically not used for ultra-low latency needs because their high throughput comes at the cost of batching and scheduling delays; a GPU might need tens or hundreds of microseconds to dispatch a kernel, which is far too slow for trade signal generation on a per-tick basis.
* **Throughput and Parallelism:** In scenarios where throughput (operations per second) is crucial, FPGAs and GPUs both offer massive parallelism, but of different flavors. GPUs can apply thousands of cores to *data-parallel* problems (e.g., risk simulations or option pricing across many scenarios), which can yield high throughput for batch computations. FPGAs, however, shine in streaming throughput – they can process multiple data streams or pipeline stages concurrently without contention. For market data processing, an FPGA can handle *multiple exchange feeds simultaneously at line-rate* (10 Gbps or 25 Gbps links), something that would tax a CPU as traffic scales. The example of an FPGA system handling 150,000 orders per second demonstrates high throughput on transactional workloads. Another aspect is that FPGAs can be designed to **scale with more logic rather than slower clocks** – e.g., duplicating an indicator calculation module 10× on the FPGA to handle 10 symbols in parallel, all updating at the same time. A CPU would time-share one core across those 10 symbols (serializing the work), and even a GPU would need to batch them and might incur latency for kernel launches. Thus, for continuous real-time processing of many independent signals (as in monitoring hundreds of stocks for triggers), FPGAs offer a combination of high throughput *and* low per-signal latency.
* **Power Efficiency:** FPGAs often deliver better performance-per-watt for specialized tasks than general CPUs/GPUs. By customizing the logic to do exactly what is needed (and nothing more), they avoid the energy overhead of fetching instructions or running unused circuits. Studies and industry reports consistently note that *FPGAs consume significantly less power than GPUs or CPUs for the same workload when optimized*. In trading data centers, power and cooling are limited resources, so doing more with less power is a tangible advantage. For instance, if a single FPGA card can replace an entire multi-server CPU-based ticker plant, the power usage can drop dramatically (perhaps tens of watts for the FPGA vs hundreds for multiple servers). The cited FPGA trading engine processing 150k orders/second not only was faster, but explicitly *consumed considerably less power than its CPU-based counterpart*. On the GPU side, while GPUs excel at bulk computation, they tend to draw high power (hundreds of watts) and are usually underutilized in low-latency scenarios (where work is not enough to occupy thousands of threads continuously). Therefore, an FPGA solution can be both *greener* and cheaper to operate for continuous trading workloads. One trade-off is development effort – historically, achieving those power-efficient designs required significant engineering time, which is a “human cost” not reflected in runtime metrics. However, with better design tools and pre-optimized IP, the energy advantage of FPGAs is increasingly accessible.

In aggregate, **FPGAs provide unmatched low latency and competitive throughput with lower power draw** for the specific pattern of tasks seen in financial trading. CPUs remain important for their flexibility and ease of development – they might handle control, rare complex events, or simply run strategies that don’t demand extreme speed. GPUs find use in offline analysis, Monte Carlo simulations for risk, or machine learning model training – areas where *throughput* is paramount and latency of individual operations is secondary. But for real-time signal processing on streaming market data, FPGAs have proven to outperform, enabling what one vendor calls “performance in nanoseconds, not microseconds”. This performance gap is the reason why many trading firms, especially those in HFT, consider FPGA acceleration a necessity for the most time-critical parts of their infrastructure.

## **Bridging Academia and Industry: Aligning an FPGA Project with Real-World Standards**

Our project FPGA implementation of technical analysis indicators, provides a valuable hands-on experience but often in a simplified environment. Bridging the gap to industry standards involves recognizing and addressing several key differences and adding certain features to align with real-world expectations:

* **Data Feeds and Interfaces:** In an academic setting, one might use static datasets or simple input scripts to feed price data into the FPGA design. In industry, trading systems interface with live market data feeds (e.g., NASDAQ’s ITCH feed, Bloomberg B-Pipe, etc.) that come over Ethernet with specific protocols. Aligning with industry means incorporating standard *feed handler interfaces* – for example, parsing a market data protocol in hardware or ensuring the FPGA can ingest data via a network interface or a high-speed bus. An academic project could evolve by integrating an Ethernet MAC and a lightweight parser for a chosen feed format, so that the indicator calculation module can receive data in real time. This would simulate a true trading environment where tick data arrives continuously. Even if full feed handling logic is beyond scope, using realistic data formats and perhaps replaying recorded market data through the FPGA (via a PCAP file and test bench) can demonstrate industry relevance.
* **Precision and Data Types:** Industry trading systems often use fixed-point arithmetic or carefully managed floating-point (sometimes custom data types) for speed and determinism. An academic project might initially use simple integer arithmetic for indicators (e.g., prices scaled by a factor). To align with industry, one should consider the precision needs – for example, computing an RSI might require decimal precision to match what traders expect. FPGA designs could be enhanced to use *fixed-point representations with adequate fractional bits* to represent financial numbers (prices, averages, etc.). This ensures that the output of the FPGA indicators matches the accuracy of software indicators used in practice. It also involves validating that overflow/underflow are handled (e.g., using larger bit-width accumulators for summations). Industry standards might also dictate certain data formats (32-bit fixed point, IEEE 754 floats, etc.), so ensuring compatibility there is useful.
* **Latency and Throughput Metrics:** Academia might emphasize functional correctness and some micro-benchmarks, but industry demands strict quantification of latency (often the *99.99th percentile latency* under load) and throughput (e.g., max messages per second). Aligning the project with this requires rigorous testing: measuring how many ticks per second the FPGA design can handle, and what the processing delay per tick is. This could involve synthesizing the design on actual hardware and using an oscilloscope or logic analyzer to measure timing, or simulation with timing annotations. For example, demonstrating that the FPGA RSI calculator can sustain, say, 10 million updates per second with a fixed 50 ns latency would speak in the metrics language that industry cares about. Additionally, considering *scalability* (can the design handle more symbols by instantiating multiple parallel modules? How does it scale on a larger FPGA device?) connects to real trading system requirements.
* **Reliability and Fail-safe Design:** Trading firms operate in an environment where downtime or misbehavior can be very costly. Thus, professional FPGA trading systems include features for reliability – such as watchdog timers, fail-safe modes that revert to software if hardware fails, and thorough verification. Academic projects can take a step in this direction by adding extensive verification (beyond basic testbenches). For instance, comparing the FPGA output of an indicator to a software reference for a long sequence of random test data to ensure no divergence, or testing boundary conditions (maximum/minimum prices, abrupt jumps, etc.). While a full production-ready failover system is too complex for a student project, awareness of these concerns and design choices like using synchronous resets, avoiding metastability on asynchronous inputs, and documentation of design limitations all contribute to a more industry-aligned project outcome.
* **Integration with Strategy Logic:** In practice, an indicator like RSI or MACD is just one part of a larger trading strategy. Industry systems would take the output of these indicators and feed them into decision logic (maybe combining multiple indicators or applying rules to generate orders). The academic project can emulate this by implementing a simple **trading decision module** that acts on the indicator signals – for example, generating a “buy” signal if RSI falls below a threshold and price is above a moving average. Indeed, the repository already includes a *trading decision logic based on technical indicators*. Expanding that to a more sophisticated strategy (even if not profitable per se) is a good exercise. It aligns with industry practice of end-to-end system development: feed → indicator → decision → output. Additionally, industry strategies must handle *multiple instruments and timeframes*. The project could be extended to compute indicators on multiple data streams (say, stocks A, B, C simultaneously) or maintain multiple window lengths (e.g., both a 50-period and 200-period moving average). This tests the modularity and resource usage of the design, much like an FPGA trading engine tracking many symbols.
* **Use of Standard IP and Cores:** To meet industry standards, one often leverages proven IP components (for memory interfaces, DSP, networking). An academic project might start with custom FIFO designs or ad-hoc state machines. Refactoring it to use, for example, a FIFO from the FPGA vendor’s library for price storage, or a DSP block for multiplications, can improve reliability and clarity. It also familiarizes the student with tools and cores actually used in companies. For instance, using the Xilinx AXI Stream interface protocol between modules would make the design more portable and comprehensible to others in the field (since AXI Stream is a common standard for data flow in FPGA designs). Adopting such standards in the project code structure (even if not strictly required to get it working) is a step toward industry convention.
* **Collaboration and Documentation:** In industry, FPGA projects are developed by teams and thus require clear documentation (for future maintainers) and often collaboration between software and hardware engineers. Treating the academic project with similar rigor adds value – writing a concise specification of the indicator algorithm, the assumptions made (e.g., input data rate, data width), and documenting the hardware module interfaces and latency. This not only improves the report for academic evaluation but also mimics the design documentation one would produce in a company. Moreover, demonstrating how the hardware could interface with a software component (for example, a C program on an embedded ARM that sends data to the FPGA and reads signals back) shows an understanding of system-level integration.

By addressing these areas, an academic FPGA project evolves from a proof-of-concept into a prototype that embodies **industry best practices**. The goal is to ensure that the skills and design developed academically are transferable to real trading system development. For instance, a project that initially “hard-codes” an RSI for one stock with a fixed 14-period length can be generalized to handle a parameterized period and multiple stocks via generics or vectorized logic – an approach much closer to how a product would be built for broader use. Aligning with industry standards also makes the work more publishable and demonstrable, as discussed next.

## **Roadmap for Further Development and Impact**

The current project – implementing technical analysis algorithms on FPGA – can be steered in two complementary directions: one towards **academic publication** and the other towards a **product prototype** for industry. We provide recommendations for each path:

### **Towards a Publishable Research Paper**

To turn this project into a publishable academic paper, consider the following steps and angles:

* **Identify a Clear Novelty or Contribution:** Survey existing literature (some of which we’ve cited) to pinpoint what aspect of your FPGA implementation is unique or extends knowledge. It could be *implementing a technical indicator in hardware that hasn’t been done before*, or a new optimization technique (for example, a novel pipelining method for the moving average window, or a resource-efficient way to combine multiple indicators). Emphasize any such novelty in the paper’s introduction as the main contribution.
* **Structured Paper Outline:** A typical academic paper would include:  
  + **Introduction:** State the problem (need for low-latency technical analysis), its significance in finance, and summarize your solution (FPGA-based accelerator) and results (e.g., latency achieved, speedup over CPU).
  + **Background and Related Work:** Discuss prior works – e.g., mention HFT FPGA use broadly, and specifically note that while FPGAs are used in trading, *little prior work addresses technical indicators in hardware*. Cite works like Ali et al. 2024, etc., to show the context. This section suits your work relative to both industry practice and academic research.
  + **Methodology:** Describe your FPGA design in detail. This should include the algorithmic formulas for RSI, EMA, etc., and how they are mapped to hardware (state machines, pipelines, fixed-point arithmetic, data flow between modules). Diagrams of your system architecture (e.g., a block diagram of modules: FIFO, computation unit, decision unit) can greatly aid understanding. Mention any optimizations (like using a sliding window technique to avoid recomputation, as you did, or the FSM control for sequential operations) – those are valuable engineering insights for readers.
  + **Hardware Platform and Implementation:** Specify the FPGA platform (e.g., Xilinx Kintex-7 or Zynq, etc.), clock speed, resource utilization, and how inputs/outputs are handled. If you did co-simulation with a CPU (hybrid system), describe that partitioning. Essentially, this section should allow someone else to reproduce or at least understand the implementation specifics.
  + **Results:** Provide quantitative results. For a paper, it’s crucial to have comparisons – e.g., latency of hardware vs software for the same task, resource usage vs. performance trade-offs, perhaps even FPGA vs GPU if applicable. If possible, measure the throughput (max updates per second) and latency (perhaps using simulation timestamps or on-board timers). Also include the effect of adding more indicators: does combining RSI and SMA in parallel double resource use but still meet timing, etc. Graphs and tables will be useful (for example, a bar chart of latency: CPU vs FPGA for RSI calculation, or a table of resource usage for different window sizes). If you have access to tools like Xilinx Vivado’s performance analysis, you might report the FPGA’s post-route timing (e.g., the design runs at 200 MHz clock, which implies a base 5 ns tick processing).
  + **Discussion:** Interpret the results. Highlight, for instance, that *FPGA achieved deterministic 1 µs latency for RSI on live tick data, versus 50 µs for an optimized C implementation on an Intel i7, a 50× speedup*. Discuss what this means for trading – e.g., such acceleration could enable strategies that react faster or handle more data. Also discuss limitations: perhaps the FPGA implementation supports only a certain maximum data rate or the design needs additional features (like dynamic parameter update) for real-world use. This shows awareness and sets up future work.
  + **Conclusion and Future Work:** Sum up the work and suggest next steps. Future work can include supporting more indicators (MACD, Bollinger Bands, etc.), moving to a larger FPGA for more parallelism, or integrating a complete trading strategy. Connect it back to the broader significance: e.g., “This work demonstrates the viability of hardware-accelerated technical analysis for low-latency trading, opening avenues for further research in reconfigurable trading systems.”
* **Select an Appropriate Venue:** Given the interdisciplinary nature (finance and reconfigurable computing), you have options:  
  + *Reconfigurable Computing Conferences:* **FPL (Field-Programmable Logic and Applications)** or **FCCM (Field-Programmable Custom Computing Machines)** often have sessions on accelerated finance or case studies of FPGA applications. Your project would fit well as a case study of FPGA for finance. **IEEE ReConFig** (International Conference on Reconfigurable Computing and FPGAs) is another, possibly with a bit lower barrier, where many applied FPGA projects are published.
  + *Journals:* **Journal of Signal Processing Systems** or **ACM Transactions on Reconfigurable Technology and Systems (TRETS)** could be suitable for a longer-form article. Additionally, given the focus, even a **journal in finance technology** (like *Journal of Finance and Data Science* or *IEEE Journal on Emerging Topics in Computing* if they have a special issue on hardware acceleration) might be viable. There are also open-access journals such as **IEEE Access** where practical implementations are welcome, though these require article processing fees.
  + *Student/Regional Conferences:* If aiming a bit lower, a university-level conference or a regional IEEE conference could be a stepping stone. However, considering the novelty, targeting an international conference like FPL or a workshop at a larger conference (e.g., a workshop on high-performance computational finance) could give the work good exposure.
* **Angle the Paper’s Contribution:** For instance, the contribution could be framed as *“An FPGA-Based Technical Analysis Engine for Low-Latency Trading”*. Emphasize that this is (presumably) the first implementation of a complete technical analysis pipeline (from data input to trading signal) on FPGA at the undergraduate level, achieving X latency. If you discover any interesting trade-offs (maybe how much FPGA resources are needed for a given indicator accuracy, or how different indicators compare in hardware efficiency), highlight those as findings. The paper should not just be “we implemented it,” but also *“what we learned”* – e.g., “RSI is multiplication-heavy but easy to pipeline, whereas moving average is addition-heavy and memory-bound; accordingly, our FPGA design handled them with different optimization strategies, which could inform future FPGA financial computations.”
* **Cite Relevant Work Properly:** To satisfy academic rigor, cite all relevant prior works (including those by vendors if they provided technical whitepapers). For example, cite the Ali et al. paper for similar work, cite any HFT-FPGA papers for context, and even cite a vendor whitepaper if it has technical merit (e.g., Xilinx’s solution brief could be cited to say “Xilinx provided an open reference for trading algorithms, but our work focuses specifically on technical indicator acceleration.”). Demonstrating awareness of both academic and industry literature will strengthen the paper.

In terms of venue-specific formatting and reviews: expect that reviewers will look for clarity in how the algorithm maps to hardware and evidence that the design was actually synthesized/tested, not just theoretical. So including that the design was run on FPGA board XYZ and processed real market data sample will add credibility. Also, if possible, share that the code is open-source (e.g., via a GitHub link) – this transparency is often appreciated in academic circles and allows others to build on your work. By following these steps, the project can be transformed into a solid research publication that contributes to the small but growing body of knowledge on reconfigurable computing in finance.

### **Towards an Industry-Ready Prototype**

Concurrently with academic publication, you can develop the project as a **prototype for a product or startup idea** in the fintech hardware space. Here are recommendations for technical and demonstrative next steps:

* **Extend to a Complete System Demo:** Move beyond isolated indicator computation to a full *demo trading system*. This could be a simplified setup where the FPGA reads a live or recorded price feed, computes an indicator, makes a trade decision, and sends an output (for example, logs a trade action or lights an LED to indicate “buy” or “sell”). A concrete demonstration: use a PC or embedded CPU to stream market data (perhaps from a CSV file of historical tick data) into the FPGA in real time (e.g., via UART or Ethernet if available on your FPGA board). The FPGA processes the data each tick, and when a certain indicator condition is met, it triggers a simulated trade message back to the PC. The round-trip can be measured to show latency. Packaging this as a video or live demo where one can see, say, a rolling RSI being calculated on an HDMI display or the decisions being made, will showcase the practicality of the prototype. Essentially, turn the project into something that *looks and behaves like a mini trading system*, not just separate pieces.
* **Use Real Market Data for Testing:** To convince industry stakeholders, test your FPGA system on *realistic data*. You can obtain historical intraday data for a stock or cryptocurrency (many exchanges provide sample data or there are public datasets). For instance, get one day of tick data for a highly liquid stock and run it through your FPGA indicator calculator. This will test the system’s ability to handle the data volume and also allow you to validate that the FPGA’s outputs match known indicator values. If possible, use data that includes a volatile period (so your indicator signals actually trigger some events). Presenting results such as “On historical data from NASDAQ on Oct 10, 2023, our FPGA system generated buy/sell signals that align with an RSI(14) threshold strategy, and it did so with sub-microsecond processing latency per tick” makes a compelling case. If you can compare this to a software running the same data, that would further illustrate the value (e.g., software might lag ticks or use more time).
* **Scalability and Integration Considerations:** Think about what it would take to integrate your FPGA module into an actual trading platform. For instance, how would it connect to an order execution system? Perhaps define an output interface in your design that formats the trading signal into a message (it could be as simple as writing to a memory-mapped register that software could poll, or sending a packet). By defining this interface, you make the prototype more concrete. Additionally, consider scalability: if an exchange feed has thousands of symbols, an FPGA product would need to track many of them. You might not implement thousands, but you could parameterize your design to handle N symbols and demonstrate it for N=4 or 8 as a proof. This shows that the architecture could be instantiated multiple times on a larger FPGA. Document or demonstrate how resource usage scales with number of indicators or symbols – this is something a product manager would ask about (can we support more instruments by using a bigger FPGA, or by time-multiplexing? etc.).
* **Optimize and Profile:** In a product context, you might need to optimize for specific metrics, whether it’s fitting in a smaller FPGA to cut cost, or maximizing clock speed for lower latency. Use FPGA vendor tools (timing analyzer, power estimator) to identify bottlenecks. For example, if your design’s critical path is the computation of a division for RSI, consider using a pipeline or an approximate method to improve timing. If resource usage (LUTs/FFs) is high, see if there are areas to simplify (maybe the trading decision logic can be microcoded rather than fully unrolled). The goal would be to ensure the design can run at a high clock rate reliably and possibly to quantify the **power consumption**. If you have access to an FPGA board with power measurements, showing that your FPGA uses only, say, 5 watts to do this real-time analysis can be a selling point (especially compared to a general-purpose system). These optimizations move the project from a proof-of-concept closer to a deployable engine.
* **User Interface and Control:** Develop a simple interface to control the FPGA logic at runtime. For instance, a software GUI or console where a user can change the indicator parameters (window length, thresholds) and the changes reflect on FPGA calculations. On many FPGA platforms, you can use memory-mapped I/O or an AXI bus to let a CPU core update registers on the FPGA. Implementing this would allow, say, changing the period of the moving average without re-synthesizing the FPGA – a critical feature in a real product (traders want to tweak strategy parameters on the fly). Even if your board is not an SoC, you could simulate this by having a block of BRAM that serves as configuration and can be edited via JTAG or some interface. The presence of a *control plane* in your design (for configuration) and a *data plane* (for the streaming data) is very much in line with professional system design. It also makes the demo interactive – a user could try different indicator settings and see the FPGA respond.
* **Consider Hardware Platform for Deployment:** If this were to be a product, on what hardware would it run? Perhaps on an FPGA accelerator card that plugs into a PC, or an embedded FPGA in a network adapter. For demonstration, if you used a common development board (like a Xilinx ZedBoard or an Alveo card if available), you could highlight that. If not, conceptually suggest the deployment: e.g., “the design could be loaded onto a PCIe FPGA card co-located with a trading server to act as a low-latency signal co-processor.” This helps frame the prototype in a real-world setting. For instance, you might outline a system diagram where market data arrives into a PC, is forwarded to the FPGA card for processing, and the FPGA signals trigger an order via the PC’s trading software. Such a diagram or description in documentation will help stakeholders visualize integration.
* **Next Technical Steps:** Beyond the prototype, chart out a roadmap. Perhaps the next steps are to incorporate more complex indicators like MACD (which you can derive since you have EMA modules) or to implement a small portfolio of strategies on FPGA. Maybe integrating simple risk checks (like “don’t buy more than X units in a second”) in hardware to show the ability to do pre-trade risk at line speed. From a systems perspective, another step could be connecting two FPGA boards to simulate a multi-node system (one as market data publisher, one as consumer) to test end-to-end latency with actual serialization and deserialization of data. Listing these future enhancements shows a path to a fuller product and can be persuasive for, say, a grant or an investor who might be interested in a hardware-accelerated trading solution.
* **Evaluation and Feedback (Industry Mentors):** If possible, present the prototype to someone in industry (perhaps an FPGA engineer at a trading firm or a fintech startup) to get feedback. They might point out practical considerations you hadn’t considered or validate that the direction is promising. Even incorporating hypothetical requirements from an exchange (like handling bursts of 50,000 messages per second, or complying with a specific protocol) can drive the prototype closer to a usable product.

By following these steps, the academic project can evolve into a demonstrable **hardware trading accelerator**. Imagine having a demo where a laptop is running a trading GUI, and alongside is an FPGA board that is clearly crunching live market data and sending signals – this would clearly illustrate the concept of hardware-accelerated trading to any audience. The key for productization is robustness and ease of use: thus, focusing on things like dynamic configurability, support for multiple instruments, and clear performance metrics will make the prototype attractive. Finally, consider the end-user (trader or strategy developer) perspective: packaging the FPGA functionality behind an API (perhaps a C++ API that the strategy developer calls to get indicator values from FPGA) can also be a selling point, similar to how commercial solutions provide a clean interface to their FPGA’s capabilities.

Both the publishable paper and the prototype development share common ground in solidifying the work and demonstrating its value. Pursuing both in parallel is synergistic – the rigor required for the paper (measurements, comparisons, clear descriptions) will improve the prototype, and the practical considerations for the prototype (real data, integration) will enrich the paper.

## **Conclusion**

FPGAs have established themselves as a **critical technology in the finance industry**, enabling trading operations to reach performance levels unattainable with software-only approaches. Our exploration covered how FPGAs are deployed for ultra-low latency trading – handling tick data in nanoseconds, executing strategies in hardware, and delivering deterministic, high-throughput performance. We delved into the specific use of FPGAs for accelerating technical analysis indicators like RSI, MACD, and moving averages, showing that even these “classic” trading tools can benefit enormously from hardware implementation when applied to high-speed trading on live data. A review of the landscape identified leading commercial solutions (from NovaSparks’ microsecond feed handlers to AMD’s nanosecond trading cards) and open-source and academic efforts pushing this frontier. Benchmarks indicate that FPGA acceleration offers superior latency and energy efficiency for streaming financial computations, aligning with the needs of HFT and real-time analytics, whereas CPUs and GPUs play complementary roles for less latency-critical tasks.

For academic practitioners and students, such as those involved in this project under Prof. Vipin’s mentorship, the message is inspiring: there is a clear pathway for academic work to contribute to and even shape industry practices. By adopting industry-like design principles and thoroughly evaluating performance, an academic FPGA project on technical analysis can both be published in reputable forums and serve as a prototype for tangible fintech innovation. The recommendations provided aim to turn the project into a polished piece of research – fit for conferences like FPL or journals – and simultaneously into a prototype that could catch the eye of trading technology firms or incubators.

In essence, this deep research underscores the synergy between **reconfigurable computing and financial algorithms**. FPGAs offer a form of “hardware agility” – much like markets themselves, they can adapt (via reprogramming) and respond at hardware speed. As markets continue to evolve and volumes increase, the demand for faster and more efficient processing grows. The technical analysis accelerator project is a timely example of how leveraging reconfigurable hardware can meet these demands. By continuing to refine such projects, validating them against real-world scenarios, and communicating the results through academic and professional channels, we bridge the gap between classroom theory and trading floor deployment. This not only validates the educational endeavor but also contributes to the broader pursuit of cutting-edge **financial technology** where FPGAs are poised to remain a key player.

**Sources:**

1. Ali, A. *et al.* (2024). *“Speed vs. efficiency: A framework for high-frequency trading algorithms on FPGA using Zynq SoC platform.”* Alexandria Engineering Journal, 96, pp.1–14. (Hardware MACD, RSI implementations and 30× speedup)
2. Magmio Trading Platform – *FPGA-based system for ultra-low latency trading*, product overview (2023). (FPGA vs CPU speed advantage of up to 1000×)
3. A-Team Insight (Nov 2024). *“AMD sets gold standard, where next for ultra-low latency trading?”* (AMD Alveo UL3524 card achieves 13.9 ns tick-to-trade, 49% faster than previous 24.2 ns)
4. NovaSparks Whitepaper via A-Team (2023). (NovaTick FPGA ticker plant normalizes full market data in <1 µs; pure FPGA tick-to-trade solutions)
5. Velvetech Blog (2021). *“FPGA in High-Frequency Trading – In Pursuit of Ultra-Low Latency.”* (FPGA parallelism and determinism for predictable low latency)
6. Pure Storage Blog (2023). *“GPUs vs. FPGAs: What’s the Difference?”* (FPGAs often outshine GPUs in terms of latency and power efficiency for specialized tasks)
7. Reddit r/algotrading discussion (2024) via Lucas M. Calderon. (FPGAs consume significantly less power than GPUs/CPUs when optimized for specific tasks – operational cost benefits)
8. Gupta, D. *et al.* (Dec 2024). *“FPGA for High-Frequency Trading: Reducing Latency in Financial Systems.”* 3rd Int’l Conf. on Automation, Computing and Renewable Systems. (150k orders/sec on FPGA with much lower power than CPU)
9. Klaisoongnoen, M. *et al.* (2021). *“I feel the need for speed: Exploiting latest generation FPGAs in HFT.”* Proc. ACM HEART’21. (Real-time transformations in FPGA are simplistic due to tight time windows, but new FPGA tech allows more complex tasks)
10. Xilinx (AMD) Solution Brief (2021). *“Accelerated Algorithmic Trading (AAT) Reference Design.”* (Open-source HLS reference design for trading applications on Alveo, to reduce development time)
11. Project Repository – *FPGA-Based Technical Analysis Trading System* (2025). *GitHub: hegdemanu/Implementation\_of\_Technical\_analysis\_on\_FPGA.* (Verilog implementation of SMA, RSI, etc., with FSM design, optimized for low latency)