Smart, Fast Trading Platforms Start with FPGAs

A new framework speeds application development of ultralow-latency financial systems.

by Rafeh Hulays, PhD
Vice President, Business Development
AdvancedIO Systems Inc.
rhulays@advancedio.com

Since the advent of electronic trading, a race for speed has ensued to build the fastest and smartest trading platforms. Smart and fast translates into money. The trading platform that is able to identify a trading opportunity and execute on it first will win the day. Response time has decreased from seconds, to milliseconds, to microseconds. The drive for microsecond and submicrosecond response time is simply not possible with traditional software or simple hardware architectures, a fact that is propelling the adoption of FPGA technology in ultralow-latency systems. However, programming FPGAs requires the development and migration of existing trading strategies (largely written in C and C++) into HDL machine language. This is a fundamentally different set of skills than standard C and C++ programming and requires a new investment in people, tools and time.

C-to-HDL compiler tools that port existing C code to a hardware description language (HDL) are available, and are faster than developing the code using HDL from scratch. However, the person working on porting the code must be familiar with hardware programming or must follow specific guidelines in order to generate acceptable code. Even then, the resulting code may be opaque and less efficient than code developed natively in HDL.

To reduce the risk involved in developing HDL code natively on an FPGA Ethernet card while also slashing development time, AdvancedIO has pioneered the use of FPGA frameworks for 10-Gigabit Ethernet (10GE) communications. Our expressXG development framework tool set provides the infrastructure necessary to ensure rapid deployment of financial services and allows seamless portability to the latest generation of FPGA cards. It integrates and optimizes all necessary core functionality required to develop applications, minimizing third-party licensing costs and shaving months off the development cycle. Development teams can then integrate a C-to-HDL compiler tool into the framework and thereby have a choice of options for developing different parts of the applications.

Indeed, both methods have a place within a project. Where efficiency and compact code are essential or where components are used in multiple projects, it's best to develop HDL components natively using the expressXG framework. Where time-to-market and customization are necessary, and where code is available in C, it may be desirable to use a C-to-HDL compiler tool. This approach addresses the fear barrier that many in the financial industry have with regard to FPGAs, without compromising performance.

AN OVERVIEW OF ALGORITHMIC TRADING

Trading in securities, derivatives, futures and other financial instruments invokes images of trading floors filled with hundreds of people shouting, milling about or peering intently at computer screens. The reality is that today, algorithms running on computer servers conduct more than 70 percent of trades in the United States. In the financial industry, speed matters. The company that can first detect and act upon an opportunity stands to profit handsomely. For some trading strategies, getting in the queue first wins the trade. No wonder, then, that companies go to great lengths to ensure they have an edge—even if just a microsecond—over the competition.

In an algorithmic trading system (ATS), algorithms running on high-performance computers handle trades and make decisions. Such systems use multiple techniques to minimize latency and gain a competitive edge. These techniques include co-location with the different exchanges, use of shortest-path networks and, increasingly, utilizing 10GE to achieve lower latency. The major trading companies are adopting all of these techniques to gain an advantage over the competition.

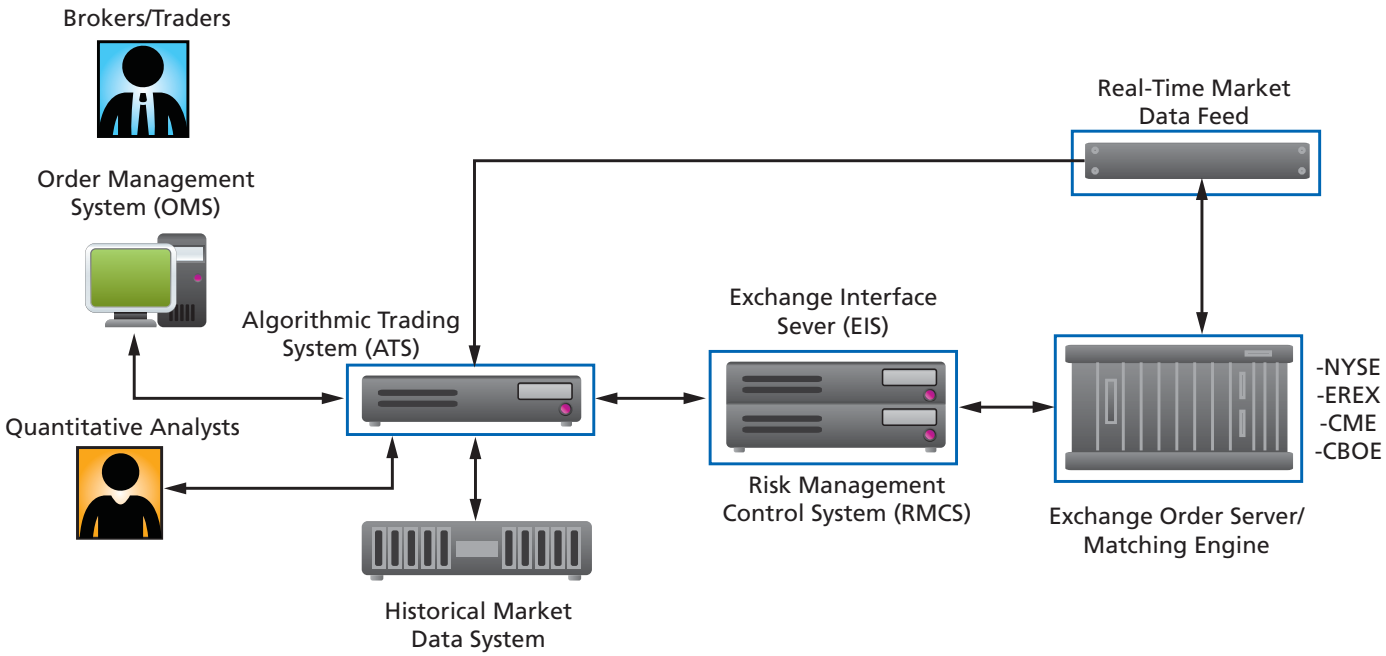

Figure 1 – A simplified diagram of a trading ecosystem. The boxes outlined in blue are points where FPGAs will significantly improve performance.

An FPGA brings a level of parallelism to financial applications that is impossible to match with general-purpose processors. Designers now are striving to pack as much functionality as possible into these FPGAs.

Trading on a stock exchange is restricted to broker-dealers and market-making firms that are members of said exchange. Some of these firms provide services to other trading companies and enable them to use Direct Market Access to the exchange order book through their accounts. Firms providing Direct Market Access must implement pretrade risk-management controls in order to limit their financial exposure and to meet regulatory requirements. To do so, they must deploy servers with software to check on all trades going through their account. This must happen at wire speed, because any delay will put them and their clients at a disadvantage. Existing software-based pretrade risk-management platforms take on the order of tens of microseconds to perform the required policy check on financial transactions. Clearly, this is not fast enough for today's traders.

Figure 1 shows the elements of a simplified trading system. In reality, an ATS may have many more elements, such as additional servers or computing clouds to perform more-sophisticated algorithms that we will not discuss here. A trading system may connect to multiple exchanges directly or through one or more exchange interface servers, electronic communications networks or other means. In addition, firewalls and intrusion-detection systems are key components of a robust trading infrastructure.

LIMITATIONS OF SOFTWARE-BASED SOLUTIONS

An ATS must be able to receive multiple market feeds, decode the different protocols used (and there are several, with colorful names like FIX, FAST, ITCH and OUCH), filter the desired trading symbols and send them to predefined "baskets" that the trading algorithms (TAs) will analyze. Once the TAs decide that a trade must be acted upon, they send a trade request to the order-management system (OMS), which communicates with the exchange interface server or with the exchanges themselves, places orders and receives confirmations. An ATS server can also stamp the incoming data with an ultra-accurate time stamp and send it for archiving. Time-stamping and market data capture can be done on either the same server (with the data going to a database server over an internal network) or a separate server.

An ATS may receive data from multiple exchanges at an aggregate burst data rate exceeding 6 Gbits/second (Gbps). Recently, Nasdaq has declared its intention to offer connectivity at 40 Gbps to its data center customers, subject to regulatory approval. This increasing rate of data is putting a significant strain on system processors and is taking away from the important task of analyzing the data and making decisions on trades. Tests on file servers using commodity 10GE network interface cards have shown that when performing TCP transfers through their native stacks, even 2.2-GHz processors can be close to 100 percent utilized just processing the IP protocols, while only achieving data rates of less than 5 Gbps. To remedy this situation, one approach is to use a piece of hardware called a TCP offload engine (TOE) to accelerate the network stack. This yields some improvement in response time.

Even with TOE being done on the network card, the amount of data that passes to the main processor is so large as to cause unacceptable delay. To reduce

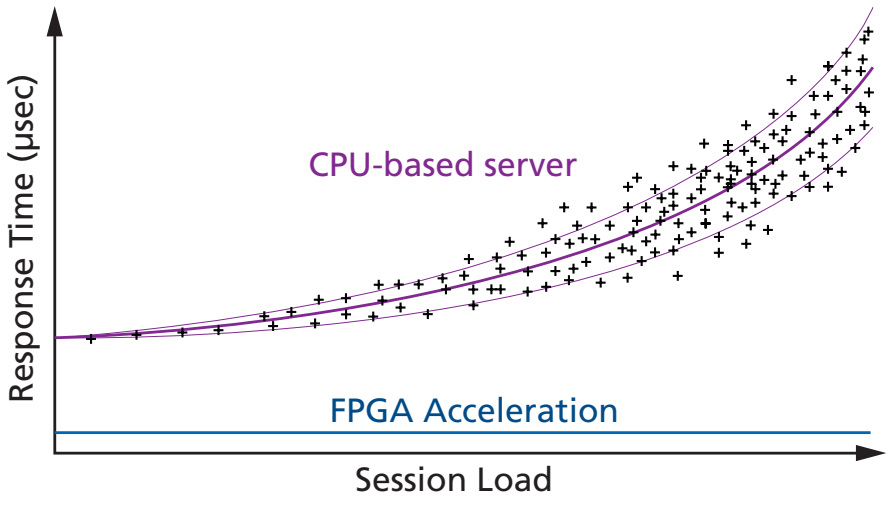

Figure 2 – Response time for applications run on a CPU-based server and those on an FPGA Ethernet card (FPGA acceleration)

latency, it is necessary to filter the relevant data at the network card before passing it to the main system processor. When all of the workload is done at the FPGA level, the response time is substantially faster. In addition, an FPGA will provide the same response time regardless of the level of load. This is not true with a regular processor. When a processor is lightly loaded, the response time is predictable, but at high load the response time increases and becomes unpredictable.

Today, communication between processors and Ethernet cards takes place via the PCI Express® bus. In theory, an eight-lane PCI Express Gen 2 bus offers a peak throughput of 4 Gbps. However, PCI Express suffers from latencies inherent in device drivers and operating-system interrupt handling. There are clear advantages to performing financial computations on the network card without having to cross the PCI Express bus to the host processor.

FPGA technology is an excellent candidate for financial applications because of its ability to process massive amounts of data in real time, with low latency and with consistency. Simply put, an FPGA brings a level of parallelism that is impossible to match with general-purpose processors. Because of the increased performance and blazing speed they are bringing to trading systems, designers are striving to pack as much functionality as possible (such as the OMS, which communicates with the exchanges, and some of the algorithms) into these FPGAs in order to reduce response time. Figure 2 illustrates the difference in response time between an FPGA-based and a CPU-based system. Not only is the FPGA considerably faster, but it also behaves consistently with increasing load, whereas the response time in a CPU-based solution increases significantly with load.

FPGA TECHNOLOGY ADVANTAGES

FPGA technology offers an unmatched capability to provide optimized solutions to the challenging problems encountered in high-bandwidth, real-time and computationally intensive financial trading applications. A solution where an FPGA executes trades at the Ethernet card has many advantages:

- Trade execution on an FPGA located close to the network's physical interface eliminates the latencies caused by the host bus, the host processor and the operating system. This drastically improves trading response time.
- FPGAs provide wire-speed performance, allowing the execution of the part of the algorithm that detects and acts on trading opportunities almost instantaneously, before others even notice the opportunity.
- FPGAs can be reprogrammed during operation, making it possible to change parameters and update algorithms to keep ahead of competitors.
- FPGAs are excellent at parallel processing, enabling them to act on multiple trades simultaneously.

Ultralow-latency financial applications are challenging to implement since they must operate at wire speed at very high bandwidth and must execute complex algorithms. Special architectures, designed with the application in mind, are required to achieve cutting-edge performance. These must take into account the need to balance computationally intensive algorithms with the need for blazing response speed to trades. Designers must make a special effort to remove bottlenecks and minimize communication and processing latencies. To avoid latencies associated with the host system bus, it is important to choose powerful FPGAs to implement the essential parts of trading algorithms, along with generous amounts of SRAM and SDRAM memory and low-latency communication ports.

Programming FPGAs using an off-the-shelf hardware module typically requires more effort and a specialized skill set than that required to create software for single or multicore processors. In addition, significant effort and investment are required to learn the documented and undocumented nuances of implementing FPGA solutions, especially at very high speeds. These characteristics may increase FPGAs' perceived risk and implementation time frames, causing some project managers and teams to avoid them and to settle instead for suboptimal software implementations. Moreover, existing trading algorithms are written almost exclusively in C and C++, and porting these into HDL code is not trivial.

There have been many approaches to simplify the task of programming FPGAs, such as embedded hard cores, software cores and tools to port C code to HDL languages. While each of these schemes has its place and each accelerates the deployment of applications, they all suffer from performance drawbacks. C-to-HDL tools are emerging as clear candidates to simplify the task of developing applications using FPGAs. However, the code these

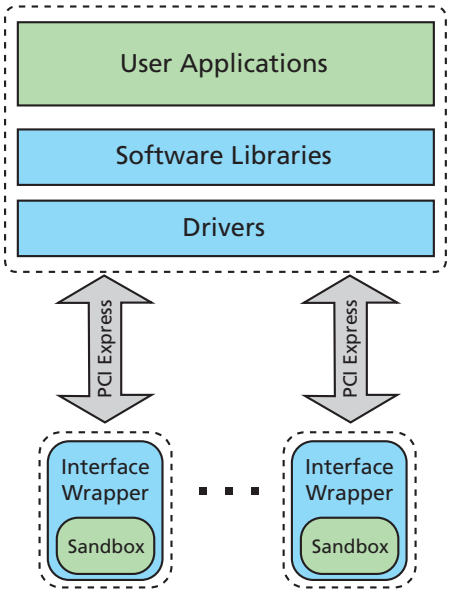

Figure 3 – A complete system-level FPGA development framework solution includes optimized low-latency Linux drivers and APIs as well as the optimized controllers shown in Figure 4.

tools generate is largely opaque and is rather difficult to optimize. Simply put, at times there is no substitute for direct HDL programming.

NATIVE HDL VS. C-TO-HDL TOOLS

A study done at the George Washington University [1, 2] reviewed the high-level language (HLL) tools that generate HDL code for FPGAs used in high-performance reconfigurable computers, and developed metrics to measure the efficiency and productivity of the various tools vs. native HDL. The researchers selected four workloads and drafted users of different levels of experience to implement the code using various tools. The results show that the HLL-to-HDL tools invariably shortened the development time by as much as 61 percent depending on the tool used. However, the results also show that the resulting code is invariably less efficient than native HDL code, with the area of utilization increasing by up to 36 percent and a frequency reduction of up to 50 percent. In addition, the research shows a considerable reduction in throughput when using some of these tools. [2] Moreover, the code these tools generated is opaque, making debugging on the HDL level challenging.

We should note, however, that the four designs used in the test were relatively simple. Some of the financial algorithms and applications are substantially more complex, and implementing them in native HDL is more challenging. We expect that as more-complex algorithms are implemented in FPGA, we will see more savings in time when using HLL-to-HDL tools. This is especially true when the algorithms are already available in C and need to be ported into HDL rather than being written in C first and then ported.

It is important to remember that C and HDL are fundamentally different, and not everything that is written in C translates well into HDL. As seen in Table 1, there are currently several variants or subsets of the C languages that vendors are using. There is an attempt under way to standardize on OpenCL, an open standard for cross-platform, parallel programming of modern processors that is being adapted for use on FPGAs. [3]

FPGA DEVELOPMENT FRAMEWORK IN HDL

Another approach to simplifying and shortening the development cycle is to use a framework written in native HDL which is highly optimized for latency and performance (Figures 3 and 4). The framework abstracts the details of Ethernet protocols and interfaces, memory controllers and host fabric interfaces, thereby reducing the development effort and schedule for designers to implement custom algorithms. This allows developers to focus 100 percent of their time on application development and integration rather than on getting all the external interfaces working on the FPGA card. A proper development framework must ensure application portability among FPGA device families and within the same family of cards. This significantly reduces the costs of future migration or upgrade cycles.

We have implemented and validated our development framework on the various families of high-performance 10GE cards from AdvancedIO Systems, which are used for multiple applications in the

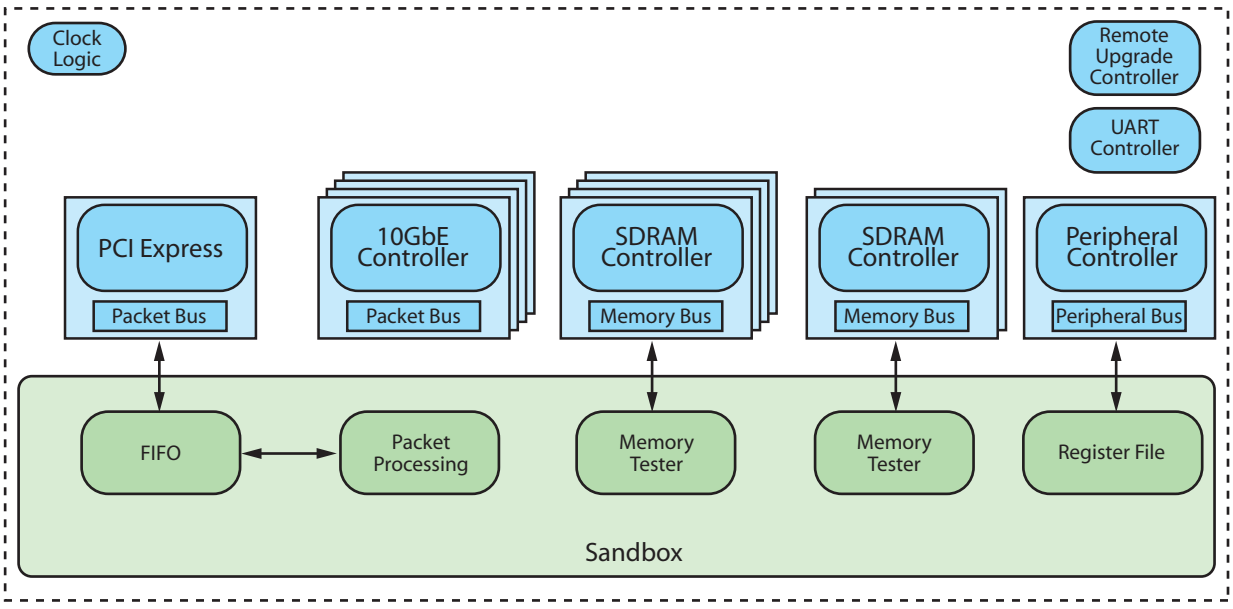

Figure 4 – A high-level view of the expressXG FPGA development framework from AdvancedIO. The top-row components (in blue) are the controllers integrated into the hardware. The components at bottom provide an infrastructure for development.

	Tool	Development Time (hrs)	Frequency (MHz)	Area (% Utilization)
1	C	3.0	–	–
2	Impulse-C	6.0	125	21.25
3	Handel-C	7.5	200	20.25
4	Carte-C	6.5	100	19.50
5	Mitrion-C	10.75	100	22.75
6	SysGen	9.0	200	19.00
7	RC Toolbox	9.0	200	19.00
8	HDLs	15.25	200	16.75

Table 1 – This comparison of HLL-to-HDL tools shows that while they deliver a considerable saving in time, the resulting code is not as efficient as that created natively in HDL. [1]

defense, financials and telecommunications markets.

The V5022 card, optimized for financial trading, features the Xilinx® Virtex®-6 HXT family of FPGAs and has all the necessary elements that an application architect needs when implementing a high-capacity, ultralow-latency trading solution. The Virtex-6 HXT family offers a large number of logic resources for complex algorithm implementations. It has two independent banks of up to 8-Gbyte, 533-MHz DDR3 SDRAM and four independent banks of up to 144-Mbit, 350-MHz QDRII+ SRAM, ideal for advanced algorithms requiring buffering or ultrafast lookup tables. The V5022 card has four 10GE ports and offers a significant reduction in latency measured from the optical cable to the MAC interface (Layer 2) inside the FPGA device.

The V5022 supports a PCI Express Gen 2 host interface. An intercard high-speed port ensures ultrafast communications between different cards in the system without the need to use the host system bus, providing a further reduction in latency. This supplies the trading platform with additional high-speed processing capability for the implementation of more-complex trading algorithms.

The development framework provides the major functionality that must be integrated into the board to ensure

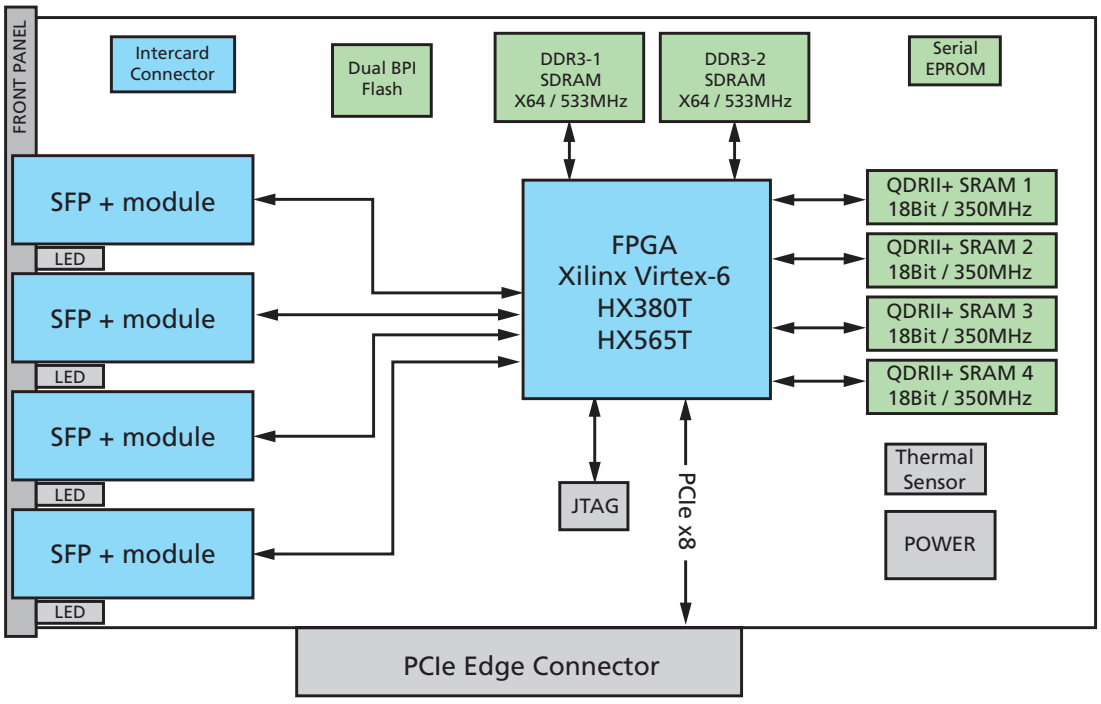

Figure 5 – A high-level diagram for the V5022 shows the different elements that an FPGA designer needs to communicate with when implementing a solution.

that everything works the way it should. Suffice it to say that the logistics of acquiring, integrating and optimizing such functionality is a costly and time-consuming proposition. Hence, the development framework offers a great value for project managers and help with time-to-market.

AdvancedIO has done extensive work to ensure that its framework is optimized to provide the best performance possible and that it occupies a small footprint and performs efficiently. This leaves the bulk of the FPGA resources available for the development of applications. For example, on the V5022 and when using the Xilinx Virtex-6 HX565T, the framework occupies less than 7.5 percent of the resources of the FPGA. This includes a PCIe® interface, four 10GE interfaces, two SDRAM controllers and four SRAM controllers.

The development framework provides a "sandbox" where programmers can develop their applications. It has easy-to-understand interfaces to the outside world, allowing the quick integration of applications on the FPGA card. Sample code and examples are provided to showcase how to exercise the interfaces and get data flowing through the system right out of the box, giving developers more confidence.

SHAVING MONTHS OFF DEVELOPMENT CYCLE

To reduce the risk involved in developing HDL code on FPGA devices and to reduce development time as well, our FPGA development framework integrates and optimizes all the controllers necessary to develop applications on the FPGA card. This shaves at least many months off the development of projects. We have also proposed the integration of a C-to-HDL compiler tool into the development framework, as studies have shown that while the code it generates is less efficient than native HDL coding, it delivers a significant reduction in development time.

Figure 6 – The V5022 is a single-slot, half-length card deployed worldwide in financial trading systems.

In real-world scenarios, there is no single approach that works best for all situations. A design team should have many tools and choices when weighing development options. Where efficiency and compact code are essential or where components are used in multiple projects, the best choice is to develop HDL components using the FPGA development framework. Where speed getting to market and customization are necessary, and where code is already available in C, it may be desirable to use a C-to-HDL compiler tool. It's best to develop fixed-function modules such as TCP/IP and UDP/IP stacks in HDL using the FPGA development framework, while algorithms that require frequent changes can make use of high-level language tools such as the Xilinx AutoESL [4] or the Impulse [5] high-level synthesis tools.

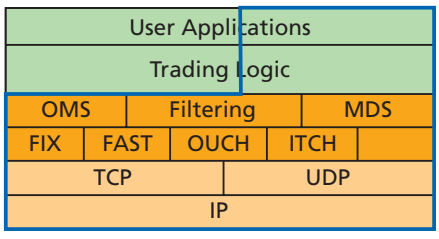

Figure 7 -- Among the elements of an ATS, the blue lines indicate the software components in which FPGAs will increase performance.

APPLICATION TO FINANCIAL TRADING SYSTEMS

Figure 7 shows the different parts that an algorithmic trading system needs to implement at a very high level. An ATS has to be able to read one or more market data feeds, perform filtering on this data against different baskets, undertake analysis, make trading decisions and communicate with one or more exchanges.

Trading logic and strategy components are subject to frequent changes and are specific to different trading companies. They are perfect candidates to be implemented using a C-to-HDL compiler in order to meet time-to-market requirements and, if the market demands, move to a more-efficient implementation at a later stage (using the FPGA development framework). On the other side, network and financial protocols do not change often and an efficient implementation of such protocols significantly affects the performance of the system. Therefore, we recommend implementing these natively in HDL using the expressXG development framework.

SPEED, RESPONSIVENESS AND PREDICTABILITY

In the financial trading sector, the drive for speed and ultralow latency is necessitating the adoption of FPGA technology. Users in this sector face many challenges, including unfamiliarity with FPGA design, different skill sets and a large existing code base in high-level languages, among others. Our expressXG FPGA development framework promises to simplify and shorten the development of applications on FPGA-powered high-performance Ethernet PCI Express cards. To assist in the porting of existing C code or where time-to-market is paramount, we propose integrating a C-to-HDL compiler within this framework.

We believe that expressXG, integrated with a C-to-HDL compiler, will assist in the adoption of FPGA technology, improving the speed, responsiveness and predictability of trading systems. For more details on the expressXG system, see *http://www.advancedio.com/*.

References

1. E. El-Araby, S.G. Merchant and T. El-Ghazawi, "A Framework for Evaluating High-Level Design Methodologies for High-Performance Reconfigurable Computers." IEEE Transactions on Parallel and Distributed Systems, *Vol. 22, No. 1., pp. 33-45, January 2011. Abstract available at* http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5445087.

2. E. El-Araby, M. Taher, M. Abouellail, T. El-Ghazawi and G.B. Newby, "Comparative Analysis of High-Level Programming for Reconfigurable Computers: Methodology and Empirical Study." Proceedings of the Third Southern Conference on Programmable Logic *(SPL '07), February 2007. Abstract available at* http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4234328.

3. Chronus Group, http://www.khronos.org/opencl/

4. Xilinx, Inc., http://www.xilinx.com/tools/autoesl.htm

5. Impulse Accelerated Technologies, http://www.impulsec.com

Security token platforms were created to make the process of token generation much simpler and more democratic. With a simplified token generation process, you should be able to avail your securities on the market within a very short period of time.Â According to them, it is highly likely that conventional assets will start shifting to more digitized ways of operation. Polymath is made up of four core layers that define the creation of tokens as well as compliance with the set operating guidelines. These includeÂ With well-defined structures and a favorable trading environment, the platform is a darling to many investors and companies looking to tokenize their securities. PDF | On Dec 1, 2017, Andrew Boutros and others published Build fast, trade fast: FPGA-based high-frequency trading using high-level synthesis.Â especially FPGA-based accelerators due to their ï¬‚exibility and. re-programmability.Â the ï¬rst comparison such that the custom trading algorithm. can start execution while the rest of the heap is being sorted. simultaneously. This also offers a scalable solution as the. Active trader community. Copy successful traders through online trading communities. Lightning fast execution. True ECN exchange model for fast order execution. EA's - Algorithmic trading. Build your own, or download Expert Advisors automated trading rules and indicators.Â Thereâ€™s a reason why Metatrader 4 (MT4) is the most widely used platform for Forex trading and FP Marketsâ€™ Metatrader 4 platform is right at the top. Our MT4 platform supports advanced charting and includes built-in technical indicators, allowing you to make better informed trading decisions. Take advantage of the MQL4 network and use any EA to automate your trading.