Run-time Reconfiguration for a Reconfigurable Algorithmic Trading Engine

Stephen Wray, Wayne Luk, Peter Pietzuch
Department of Computing, Imperial College London
London, United Kingdom
{*sjw06,wl,pp*}*@doc.ic.ac.uk*

***Abstract*—In this paper we present an analysis of using run-time reconfiguration of reconfigurable hardware to modify trading algorithms during use. This provides flexibility in algorithm design, enabling the implementation to be reactive to changes in market conditions, increasing in performance. We study what can be achieved to reduce performance loss in algorithms while reconfiguration takes place, such as buffering information during this time. Our results show our average partial reconfiguration time is 0.002091 seconds, using historic highest market data rates would result in about 5,000 messages being missed or require buffering. This is the worst case scenario, normally the system would only require a fraction of messages. The reconfiguration time is acceptable if it is under the required limit by the user to prevent business performance suffering.**

***Keywords*-Algorithm Trading; Run-Time Reconfiguration;**

I. Introduction

Algorithmic trading is a large part of the electronic trading landscape which has been evolving from the 1970s. Algorithmic trading is designed to solve problems faced by conventional trading of stock by human traders. First, an increased market volume as computers were used more frequently in communicating trades, the cost of trading dropped, which led to investors trading more frequently. This increase eclipse paratactical methods of manual management because in the 40 years leading up to 2001, the daily volume on the New York Stock Exchange increased 500 fold [1]. Second, market spreads have become tighter and latency dropped, making it more difficult for people to carry out trading of large volumes. The prevalence of algorithmic trading, the daily volume traded is still increasing, requiring faster and more powerful computers.

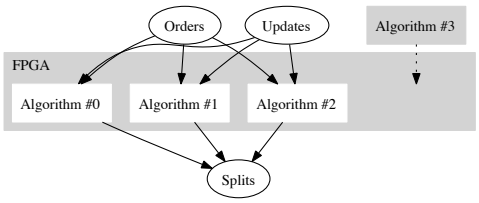
Equity trading is one individual purchasing shares from another, through a stock exchange. Buyers and sellers place orders on the exchange with a price, orders trade if the prices align. The price of a stock changes depending on the economy, news, supply and demand. An algorithmic trading engine is designed to split large orders into smaller orders before sending them to the exchange. The means the order can be executed more consistently than a human trader. The engine can react quickly to market fluctuations and handle greater volumes of orders. The engine makes its decision on received information, order parameters dictate how the order should be traded and market data updates convey the current market conditions [2]. Smaller orders don't affect the market as larger orders, in addition they conceal the activities of the investor and achieve a better overall price.

Reconfigurable hardware is a highly desirable platform for an algorithmic trading engine. Benefiting the features of reconfigurable hardware, such as low latency and low standard deviation on performance, producing a highly dependable system. This has been realised before [3], but has had little published research carried out in the area. Research has likely to have been carried out but remained unpublished because of holding a competitive advantage for the owners in the industry. One difference that needs to be addressed between software and hardware implementations is how to deal with the limited resources of FPGAs. FPGA chips are available in multiple sizes, large chips may be able to accommodate tens of algorithm instances. This increases device costs and power consumption while resources would be wasted by infrequently used components. For this reason, in this paper we will use the run-time reconfiguration feature offered by many FPGAs so that smaller, less expensive chips can be used while affording the flexibility of supporting a large number of algorithms.

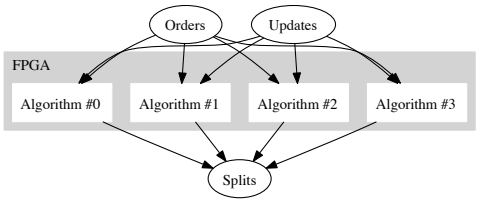
We show that the amount of time to complete a partial reconfiguration of the device changing algorithm instances is acceptable. This allows the cost of reconfiguration measured in terms of price performance to remain low. The ability for the user to deploy alternate algorithms increases the possibility that the change will compensate for any loss and by improving performance. The contributions of this paper include the following:

- An overview of why run-time reconfigurations use is attractive in an algorithmic trading engine; in Section III.
- A solution for storing instances to deploy on different market conditions; in Section IV.
- Evaluation and comparison to a related software solutions, illustrating its potential; in Sections V and VI.

This paper also covers related background work and material that motivates this research; in Section II. A comparison to a software implementation and the relation to run-time reconfiguration of hardware based systems; in Section III.

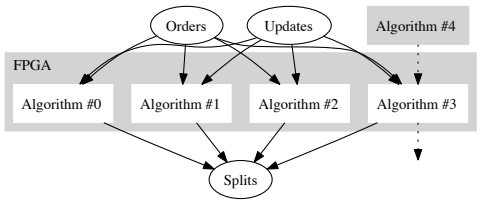


(a) Before partial reconfiguration.

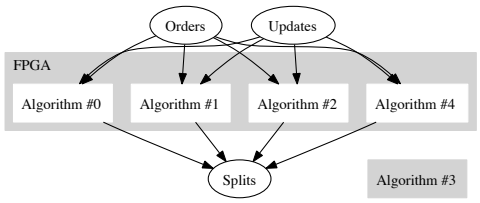


(b) After partial reconfiguration.

Figure 1. 1(a): The FPGA has space for the new algorithm to be placed. 1(b): The FPGA includes Algorithm #3, connected and processing data.



(a) Before partial reconfiguration.



(b) After partial reconfiguration.

Figure 2. 2(a): The FPGA has no space and an existing algorithm, #3 must be removed. 2(b): Algorithm #3 has been removed and replaced with #4.

II. Related Work

Inspiration for this work was previous research in related financial fields. [4] presented using FPGAs to accelerate market data processing, which is an important feature of an algorithmic trading engine. [5] shows the effectiveness of using FPGAs in financial calculations related to those performed by the algorithms. The financial media has also covered how FPGAs could be used within electronic trading. A report [6] shows that FPGAs are beginning to get market penetration where previous research has helped, shown in [4]. Article [7] explores the demand for such ability but also the problems that exist with these types of architectures.

Similar work in using run-time reconfiguration to make implementations more flexible while retaining processing ability has already been shown in [8]. This research was applied in the similar area of network packet analysis which has many similar traits to an algorithmic trading engine. [9] showed run-time reconfiguration of a software defined radio, it is not immediately relevant but many of the same problems with buffering or not buffering information during the transition period apply in this field.

III. Run-Time Reconfiguration

Figures 1 and 2 show the process of partial reconfiguration from a high-level perspective. The first scenario has the enough capacity on the device to place an additional algorithm but the second scenario doesn't have this luxury. After placement the data feeds and output are connected, processing data starts. The process doesn't intrude on the operation of any algorithms that aren't modified, they are able to continue working at full performance. The performance of algorithms implemented in hardware was explored and some limitations discovered to using this type of architecture [2]. One concern was supporting a large number of algorithms with limited resources, the available space on the FPGA chip. This paper is going to explore using run-time reconfiguration to support a large number of implementations with limited resources.

Parameters of the algorithm can be altered to allow the device to maximise its performance. On high market data update throughput it would be appropriate to change the order batch size to larger than the current number of orders being processed. If it cannot be increased enough, then increasing it to minimise wastage by making the batch size close to a factor of the total number of orders to process has a similar effect. With an increased depth of market, the update batch size can be increased to improve accuracy of decision making, using more information. There may be occasions where an algorithm could be modified to make it more effective in the short term. Alternatively one algorithm may prove popular one day, it would improve performance to increase the number of instance of the same algorithm to keep up with demand.

All of these situations lead themselves to using the run-time reconfigurable features provided by many FPGAs that allow for implementations to be modified or fully changed quickly. The disadvantage is that it takes a notable amount of time, especially when dealing with latencies in the microsecond realms. During this time, performance is reduced and trading opportunities may be lost. For this reason, we need to know how long example reconfigurations would take, during when no processing can be completed. In algorithmic trading, updates can be dropped and left unprocessed or buffered for processing later. Generally only storing the last good update is required as it contains all the current information for the current market state. If the algorithm tried to trade on updates that had expired then it may trade incorrectly, causing a detrimental effect on its business

$$\begin{pmatrix} A_{10,10} & \dots & A_{10,n} \\ \vdots & \ddots & \vdots \\ A_{m,10} & \dots & A_{m,n} \end{pmatrix}$$

Figure 3. A matrix of configurations for a single algorithm (A) varying by order (m) and update (n) batch size. Groups of matrices form a library.

performance. This is because liquidity it was targeting may no longer be available when the trade is sent. However some algorithms may require this information for other purposes such as statistics or internal state. For the time being, the algorithms that we are interested in only require the last good update when resuming processing.

A software implementation designed to load algorithms during runtime, has similarities the hardware implementation. However as previously discussed, the ability to create a new algorithm during the trading day, complete testing and compilation is impractical because of time limits. Unlike our hardware version, there is no need to compile a library of specific variations and store them separately. This is because an algorithm's memory footprint is small compared to the data that it stores and so having a large number of algorithms ready to accept information in a software based environment is possible. Memory is finite but we assume that the system has sufficient memory to operate. As each algorithm uses a quantity of time on the processing unit to execute, there is no concern over the implementation size of the algorithm and data representation size is coarsely grained. This relieves many of the problems faced by the hardware implementation. The software system can be thought of as having all the implementations ready and having to load them every time it wants to use them, much like how a reconfigurable hardware device would have to be reconfigured to use another algorithm. However for software, loading the code from memory is a natural overhead of the system and extremely quick compared to reconfiguring a hardware device. Therefore there is little benefit in designing a software system in this way, in addition the reconfigurable architecture affords many methods of increasing performance through specialisms compared to software features.

IV. Run-Time Reconfiguration Library

Since the process of compiling, testing and generating a hardware bitstream with placement information takes a significant amount of time (more than an hour), it is impractical to generate reconfigurations on the fly. Our solution is to build a library of existing implementations which have been tailored to different scenarios that can be quickly deployed to a target device. The size and granularity of the library can be determined by the user, based on a number of factors, including the scenarios that they expect to encounter and the capacity of their device. Figure 3 shows a possible set of configurations for an algorithm to be pre-generated for use during a trading day. The variables m and n are only bound by the requirements of the use and the ability for the implementation to be placed on the target device.

Table I
Size and reconfiguration time of the algorithm.

	N_{slices}	$N_{columns}$	$T_{partial}$
$A_{5,5}$	512	4	0.001394
$A_{5,10}$	538	4	0.001394
$A_{5,15}$	462	3	0.001046
$A_{5,20}$	558	4	0.001394
$A_{10,5}$	665	5	0.001743
$A_{15,5}$	641	5	0.001743
$A_{20,5}$	770	5	0.001743
$A_{10,10}$	662	5	0.001743
$A_{15,15}$	718	5	0.001743
$A_{20,20}$	737	5	0.001743

Trading activity through the trading day can be coarsely predicted by expecting higher volumes at the start and end of the trading period. In addition there is a notable bump in load during the afternoon when the USA starts trading. Future activity increases may have fore warning if a client informs the operators that they will be placing a significant number of orders. Load can be expected to rise after important news is announced which may affect the markets, the time of this may be known before the announcement. It is to be expected that during times of increased activity fewer algorithms will be used but they will provide better utilisation. During times of less activity, a wider range of algorithms could be selected to provide better trading functionality. This fine-grained approach to selecting algorithms given the current market environment, gives the user a lot of control, with performance benefits in both high and low activity periods.

V. Experimental Results

The hardware implementation was developed in Handel-C [10], which is a subset of the C language with extensions specific for FPGA development. The implementation was compiled into Electronic Design Interchange Format through Mentor Graphics DK Design Suite 5.2 before Xilinx ISE Webpack 11.1 placed and routed the target chip [2]. We use the Xilinx Vertex 5 xc5vlx30 FPGA to produce results from analysis. The xc5vlx30 is an array of 80 rows and 30 columns, totalling 4,800 available slices [11]. A total of 8,374,016 configuration bits and 6,376 configuration frames, each frame is 1,312 bits [12]. A single column contains $4800/30 = 160$ slices or 212.5 frames. The xc5vlx30's reconfiguration clock set up to 50MHz with a 16 bit data path, providing 800 Mbit/s transfer [12]. Using the above information with information from our design, we can calculate the time to reconfigure the target device under a number of scenarios. Table I shows these results from variations created from an existing algorithm.

Figure 4 shows the how the algorithm scales in three different scenarios. The first scenario varies the order batch size, this increases orders processed throughput. Increased

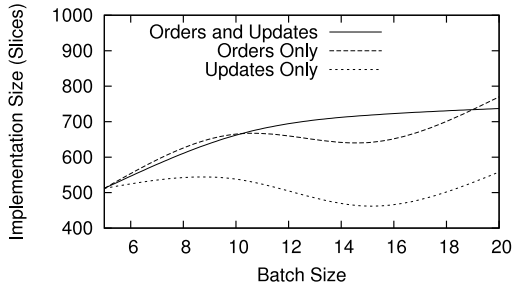


Figure 4. Size scaling graph for the participate algorithm.

in times of high market activity, or decreased to save resources. The second scenario varies the update batch size, this increases the algorithms decision making accuracy. Increased in times of large market depth, or decreased to save resources. The third scenario varies both order and update batch size. This delivers the advantages and disadvantages of both previous scenarios. The graph shows the implementation space requirements. The resource utilisation increases as the parameters increase. This is a trend as the actual utilisation varies depending on how effective the compiler is with optimising the implementation. On batch sizes which are multiples of ten there is generally a higher level of optimisation.

The device is reconfigured a column at a time, we calculate the reconfiguration time per columns for this device is 0.00035 seconds. We can now deduce how many messages would be missed or need buffering in this time. A historic maximum peak in market data information can be take as 2,526,103 messages per second recorded on February 11, 2010 [13]. This peak is the aggregate of 15 american markets and presents a worst case scenario. Using this information, we find that ˜880 messages would be missed. A system would only need fraction of these updates though. Slow stocks would normally have no updates within this reconfiguration time, busy stocks may receive a couple. Using a 32 bit data representation and an update entry batch size of 10, each update requires 640 bits of storage. This would allow 2MB of storage to hold 25,000 updates.

VI. Conclusion

We have looked at using run-time reconfiguration to change an implementation during the trading day, making it more effective and flexible. This has been an area of concern for implementing an algorithmic trading engine in reconfigurable hardware. We have shown that the device can maintain high utilisation making it highly cost effective while not loosing the ability to implement multiple complex algorithms. To show the loss is minimal, we have shown through current peak market data update rates how many messages would have been missed. This was taken from a large number of consolidated stock exchanges as a worst case scenario, where a production system would consume much less information.

We have presented a solution to reduce the time to deploy algorithms by pre-compiling and storing a selection of implementations. These can be deployed quickly given market conditions. The compilation and testing of the implementation is completed before the algorithm are required. We have also discovered that compilation optimisations carried out vary depending on the size of the implementation. Shown in our results where an increase in computation does not necessarily result in an increase in resource usage. When generating a library, it would be beneficial to remove inefficient designs to prevent their use.

Acknowledgment

We would like to thank Qiang Liu and Tim Todman at Imperial College London for their help with the Mentor Graphics DK Design Suite and Xilinx ISE. Additionally we would like to thank the three anonymous reviewers for their invaluable feedback that enabled this paper to be improved.

References

[1] NYSE, "Homepage," *http://www.nyse.com*, Mar 2010.

[2] S. Wray, W. Luk, and P. Pietzuch, "Exploring algorithmic trading in reconfigurable hardware," in *Application-specific Systems, Architectures and Processors*, July 2010.

[3] Automatedtrader.net, "FPGA's - Parallel perfection?" *Automated Trader Magazine Issue 02*, July 2006.

[4] G. Morris, D. Thomas, and W. Luk, "FPGA accelerated low-latency market data feed processing," in *High Performance Interconnects, 2009*, Aug. 2009, pp. 83–89.

[5] A. Tse, D. Thomas, and W. Luk, "Accelerating quadrature methods for option valuation," in *Field Programmable Custom Computing Machines, 2009*, April 2009, pp. 29–36.

[6] I. Schmerken, "Credit Suisse hires Celoxica for low-latency trading and market data," *FinanceTech*, Nov 2009.

[7] R. Martin, "Wall Street's quest to process data at the speed of light," *Information Week*, Apr 2007.

[8] S. Yusuf *et al.*, "Reconfigurable architecture for network flow analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 1, pp. 57–65, Jan. 2008.

[9] T. Becker, W. Luk, and P. Y. Cheung, "Parametric design for reconfigurable software-defined radio," in *ARC '09: Proceedings of the 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications*, 2009, pp. 15–26.

[10] *Handel-C Language Reference Manual*, Mentor Graphics.

[11] Xilinx, *Virtex-5 Family Overview*, Feb 2009.

[12] ——, *Virtex-5 FPGA Configuration User Guide*, Aug 2009.

[13] Market Data Peaks, "Homepage," *http://www.marketdatapeaks.com/*, Mar 2010.