Project Proposal:
Monitoring and Processing Stock Market Data In Real-Time Using the Cyclone V FPGA

Alexander Gazman (ag3529), Hang Guan (hg2388), Nathan Abrams (nca2123)

1. Introduction

The general premise of our project is to design a system where hardware and software work together to monitor real-time stock data for many stocks and implement an algorithm to analyze the market to inform a user on whether they should buy, sell, or hold the stock. The stock's fluctuations in price will also be displayed on a screen with a confidence level associated with the suggested action.

2. General Design

Our system is composed of a workstation, a FPGA development board and several peripherals (e.g. monitors, keyboards), as shown in Figure 1. In the workstation, we will mainly implement a software, that can automatically fetch stock data from Yahoo Finance API, wrap it as the payload into a TCP/UDP package, and send to our FPGA development board through Ethernet ports. In the FPGA development board, we will implement a NIOS-based system which can achieve various of functions: 1) extract the payload out of the TCP/UDP package, 2) parse the stock information, store the information efficiently (based on some sorting algorithm) into the on-chip memory for further processing, 3) display the stock movement (e.g. volume, price, etc), and more importantly, 4) implementing some data mining algorithms that can provide users information regarding the proper action towards certain stocks (e.g. whether or not should the users buy this stock at certain time).

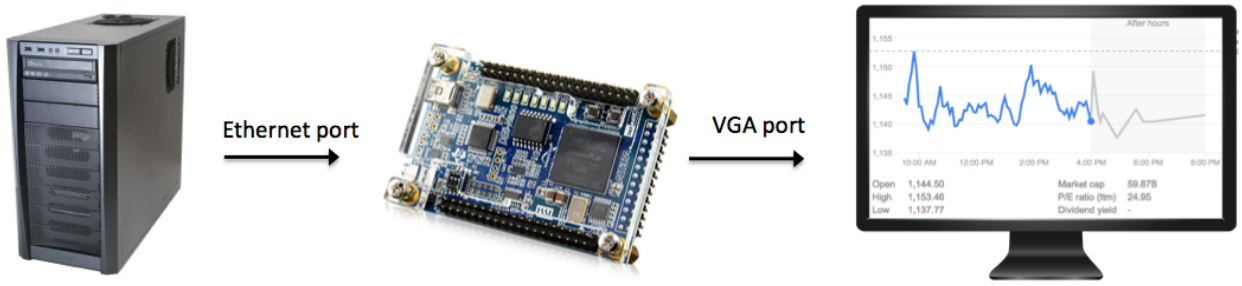

Figure 1: System composition

Figure 2 shows the diagram of our stock data processing and monitoring system. The TCP/UDP parser will first extract the internet package and get rid of the herders of the package. After that, the stock information storage will save the extracted information based on a lookup table, which contains the starting address of each stock. We are interested in exploring the possibility of using other more efficient mapping algorithms (e.g. binary tree) to improve the speed of our system. Meanwhile, the stock information storage module will not only write, but also read the stored information out of the on-chip memory based on the commands that given by the data mining module. The data mining module will run multiple algorithms (e.g. mean reversion) which can provide the users with the related indices and other metrics. The display module is needed to plot the the stock tendency and other related informations on the monitor.

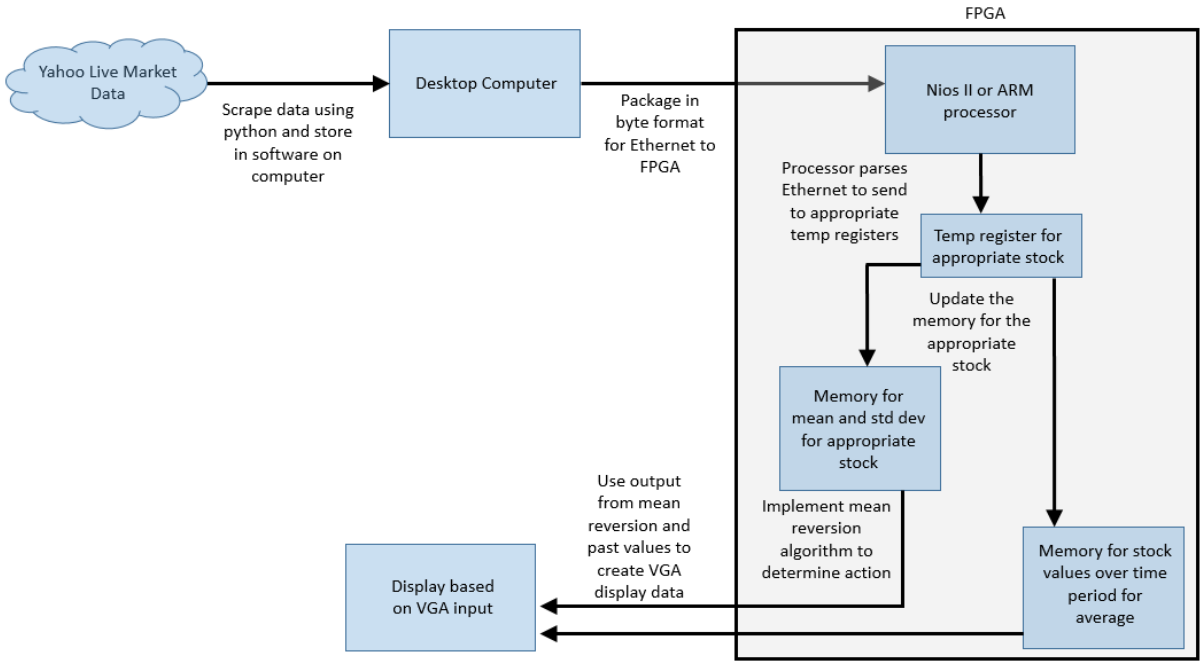

Figure 2: System block diagram

3. Inputting Stock Data

Both real time and historical stock market data will be imported from Yahoo Finance using a dedicated Python library [1]. Other data, like NASDAQ stock data are also available on the website [2]. The data is downloaded from the website based on web-scraping through Yahoo's open source API. The library allows to download for each stock symbol the following data sets: stock volume (number of buys and sells per day), opening and closing price, high and low values and the date. The information is saved in a Python dictionary data structure and parsed getting the values in interest.

To validate the code, Yahoo's stock values were plotted and compared to the online data. In figure 3, it is possible to see the correlation between the scraped and the online data of Yahoo's stock market value over time period of 4 years. The agreement between the the results further examined in TEVA, APPL and GOOGL stocks.

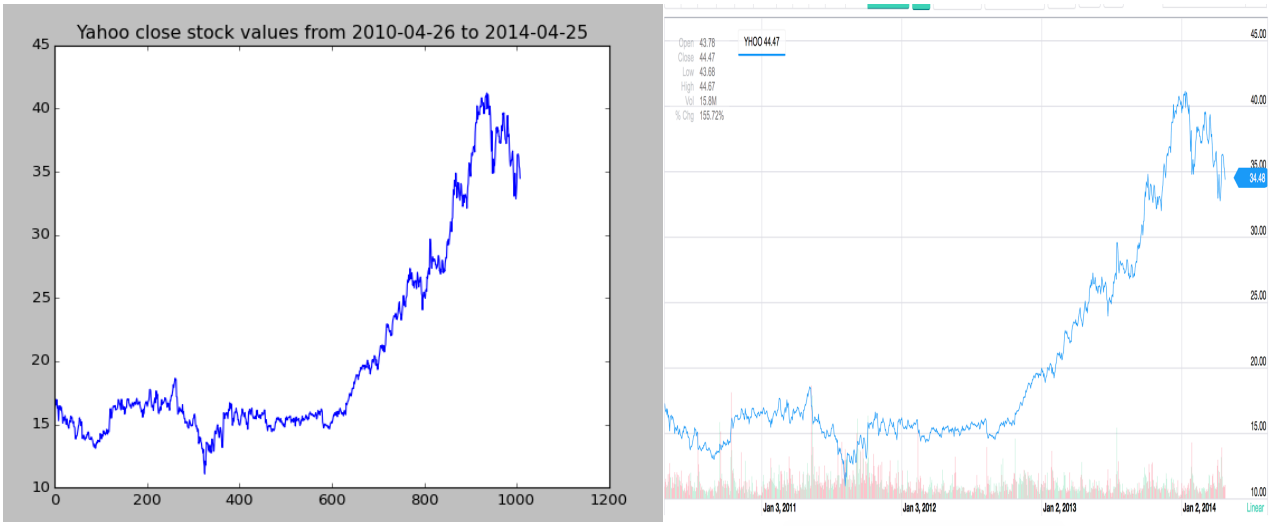

Figure 3: Comparison of Yahoo stock market data using the Python (left) versus the online data (right)

For transmitting the data to the FPGA we will design a specific hardware driver that will communicate with the software through either the ARM processor or a NIOS II processor on the board. From the given sets of information, including the real time values, the driver will accept the necessary data to evaluate the algorithm (see below).

At this point, stock market data is downloaded using Python, however, it might needed to migrate to code to C program to communicate with the hardware driver. For simplicity purposes, we can have a hybrid version of Python (embedded in Linux) and C to for downloading data and sending to the FPGA for processing.

4. Algorithm

As indicated in milestone one, one of our first tasks will be to finalize the algorithm we will be using. At this time, we believe we will be implementing an algorithm based on mean reversion. The general idea behind mean reversion is that while a stock will fluctuate, there will be some relatively steady average over time. It is expected that the stock will regress back to this mean during these fluctuations. The mean for mean reversion is often some time averaged mean over the order of 10's of days. Once the trading range of a stock is identified, the time averaged price can be computed as it relates to assets, earnings, or other metrics. The basic idea is that when the stock's price fluctuates above this mean the stock is attractive to sell and when the

stock's price fluctuates below this mean the stock is attractive to buy. Statistical parameters such as the standard deviation over the relevant time period can be used to provide a level of confidence for the decision to buy or sell (for example, only buy/sell when the stock is 0.3 standard deviations away from the mean).

At a first pass, this could be a very simple algorithm to implement, however, we expect that are more involved methods based on mean reversion that will allow us to implement a more complex algorithm (if we wish) that weighs some risk-cost into the potential earnings and use that in combination with the standard deviation to provide a suggestion to buy or sell.

While mean reversion is the frontrunner for the algorithm we will implement, other algorithms include pair trading and arbitrage. Pair trading involves two historically correlated securities. A simple example is Pepsi and Coca-cola. Since they both produce soda, we expect the stocks to be correlated. Pair trading involves taking advantage of scenarios when this correlation is weakened, shorting the over-performing stock and going long on the under-performing stock. The disadvantage of implementing this algorithm is that we don't necessarily expect real data to demonstrate a weakened correlation while we are working on the project, so we may have to introduce fake data to demonstrate that the algorithm works.

Arbitrage involves identifying when an asset does not trade at the same price on all markets. The example we found was for discrepancies between the S&P futures and the S&P 500 stocks as the S&P 500 stocks (mostly traded on the NYSE and NASDAQ) will get ahead or behind the S&P futures (mostly traded on the CME). While pair trading and arbitrage are interesting potential algorithms, we will likely be implementing some version of mean reversion as our algorithm.

5. Memory

Once we parse the stock data from yahoo and send it via ethernet to Nios II or the ARM processor, we will need to store it in memory on the FPGA. We admit that this is an area we are less certain on how to implement, and finalizing the memory design is under milestone one. If we want to be able to display data for a moving average, we will need to have snapshots stored of the time and the value of the stock. For implementing the mean reversion algorithm we will need to have the time averaged stock value and the corresponding standard deviation. Since the time sensitive data we need for the mean reversion algorithm is just the time averaged mean and the standard deviation, it may be worthwhile to split the memory into two sections for each stock: a quick updated block that only contains the mean and standard deviation, and a lower priority block for containing snapshots of the past data for when the user wants to display the stock's value over the time averaged period. New data updates the mean and standard deviation, and shifts the memory of the time averaged period, deleting the oldest data entry.

Time averaged mean price
Time averaged price standard deviation

Time[0]	Stock_Price[0]
Time[1]	Stock_Price[1]
Time[2]	Stock_Price[2]
Time[3]	Stock_Price[3]
Time[4]	Stock_Price[4]
Time[5]	Stock_Price[5]
Time[6]	Stock_Price[6]
Time[7]	Stock_Price[7]
Time[8]	Stock_Price[8]
Time[9]	Stock_Price[9]

Figure 4: Potential memory design for a single stock of interest

6. Display/Output

For the display, we plan on implementing something in the vein as shown below using VGA to a screen. A stock of interest can be selected (or the display can snap to a stock when the algorithm determines that an action--buy or sell--should be performed). The stock name will be shown, and there will be a graph of the stock's price over the time period for which the mean has been averaged over (on the order of 10's of days). On the graph, there will be a shaded "non-action interval" which can be calculated from the algorithm and can take into account the standard deviation and other risk-assessment parameters determined from the algorithm; within this interval the stock is held. If it the stock rises above this "non-action interval", the action is to sell. Similarly, if the stock falls below the "non-action interval", the decision is to buy. We could also display some stock confidence value that is based on the standard deviation and other risk assessment factors, and could normalize it to be between -1 and 1 for the "non-action interval". Finally, we will need to decide if we want to implement mimicking an outbound order byte stream in a particular protocol's format (a decision to be made by milestone one).

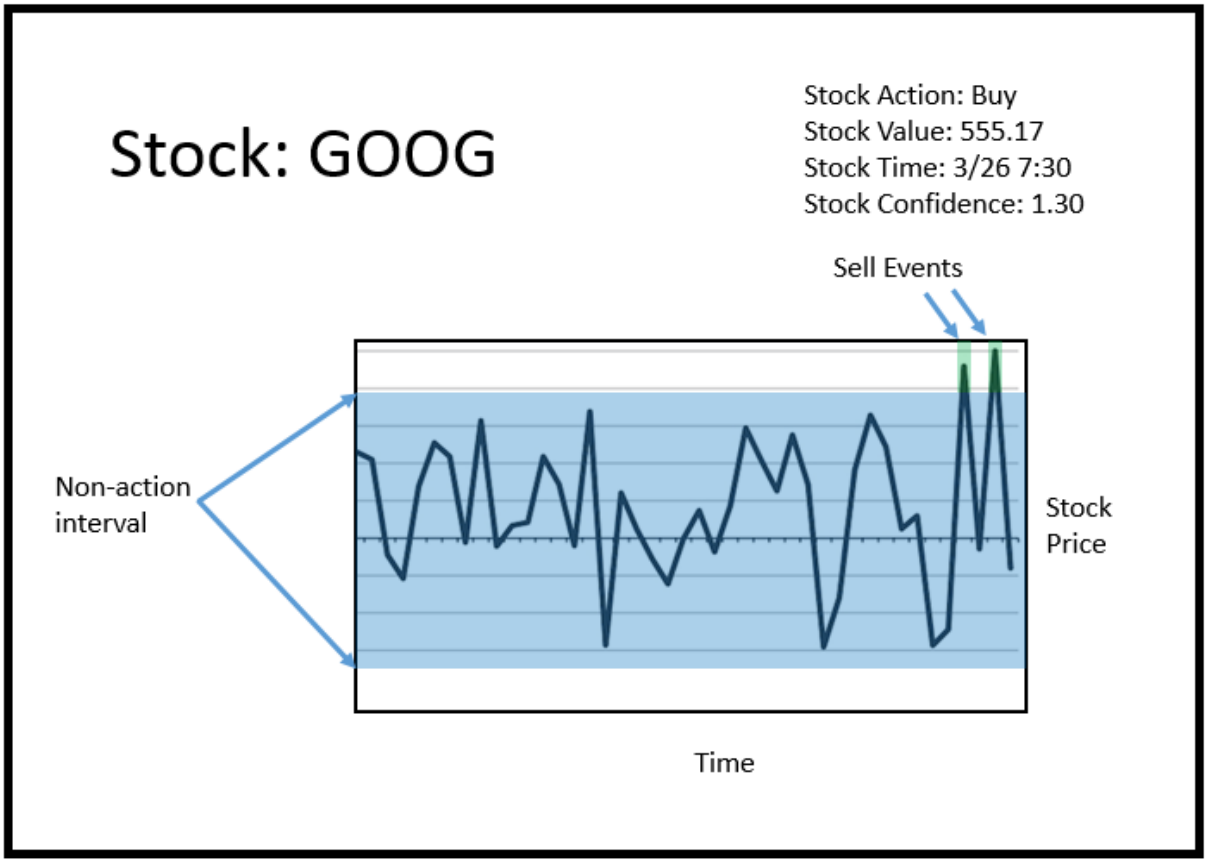

Figure 5: Potential display

6. Milestones

Milestone One (4/2/2015)

- Finalize which of the algorithms we will implement (likely mean reversion)
- Implement algorithm not in hardware (demonstrate it works in software)
- Implement parsing Yahoo stock data into FPGA (but not necessarily storing it)
- Finalize group decision on how we would like to display output, memory structure, and whether we want to implement an outbound 'order'

Milestone Two (4/14/2015)

- Implement memory structure in hardware (independent of actual data--using hardcoded values)
- Implement algorithm in hardware (independent of actual data--using hardcoded values)
- Implement first pass display output (for example, a stock's mean, standard deviation, and buy/sell/hold decision, outputted to screen)
- Implement outbound 'order' (if we decide we would like to in milestone one) using hardcoded dummy data

Milestone Three (4/28/2015)

- Connect data stream to memory structure and algorithm
- Finalize display (potentially also including a graph of the stock's price over the time period it is averaged)
- Finalize outbound 'order' with real data (if we decide we would like to in milestone one)
- Implement other FPGA to receive outbound 'order' (if time: unlikely)
- Implement additional algorithms (if time: possibly likely)

Final Project (5/14/2015)

- Hopefully just creating the presentation, however, we anticipate that the previous milestones will not go exactly according to plan (the main bullet points that concern us are implementing the memory structure and algorithm in hardware--we expect that they may spill over into milestone three).

8. Resources

1. https://github.com/lukaszbanasiak/yahoo-finance
2. http://www.nasdaqtrader.com/Trader.aspx?id=itch