Contents lists available at ScienceDirect

Alexandria Engineering Journal

journal homepage: www.elsevier.com/locate/aej

Original Article

# Speed vs. efficiency: A framework for high-frequency trading algorithms on FPGA using Zynq SoC platform

Abbas Ali [a], Abdullah Shah [a], Azaz Hassan Khan [a], Malik Umar Sharif [a], Zaka Ullah Zahid [a], Rabia Shahid [a], Tariqullah Jan [b], Mohammad Haseeb Zafar [c,*]

[a] *Department of Electrical Engineering and Computer Science, Jalozai Campus, University of Engineering and Technology, Peshawar, 25000, Pakistan*
[b] *Department of Electrical Engineering, Main Campus, University of Engineering and Technology, Peshawar, 25000, Pakistan*
[c] *Cardiff School of Technologies, Cardiff Metropolitan University, Cardiff, CF52YB, UK*

ARTICLE INFO

*Keywords:*
System-on-Chip
High-frequency trading
Xilinx Zynq-7000
HW/SW co-design
Technical indicators
Cryptocurrencies

ABSTRACT

Software-based technical indicators have been widely used for the stock market forecasting, aiming to predict market direction. Even though many algorithms for the software based technical indicators are presented, there are almost no hardware implementations reported in the literature. In this paper, the hardware implementation is presented for three commonly used technical indicators: Moving Average Convergence/Divergence (MACD), Relative Strength Index (RSI), and Aroon. Latency evaluation is conducted for Bitcoin and Ethereum within a single-day timeframe, utilizing the Xilinx Zynq-7000 programmable SoC XC7Z020-CLG484-1 platform. Additionally, various hardware/software (HW/SW) partitioning strategies are explored to leverage the flexibility of software alongside the performance advantages of hardware via the Zynq SoC platform. The results show that the best performing technical indicator is MACD with a speedup of 30 times over its software only counterpart. Furthermore, a hybrid design integrating multiple technical indicators is proposed, pairing MACD with RSI due to their competitive throughput values, differing by only 0.38 microseconds. This hybrid approach capitalizes on the parallel processing capabilities of hardware, enabling multiple systems to operate simultaneously.

## 1. Introduction and motivation

Stock market forecasting is one of the most trending problems currently faced by the financial world [1]. For successful trading automation, the entire process can be implemented in software [2,3] or hardware leveraging advanced technologies [4]. Software-based algorithmic trading offers numerous benefits, including design flexibility [5], enhanced accuracy [6], and [7] compared to human-executed trades. The researchers and developers targeting software-based methods most often aim at coming up with strategies to maximize the accuracy of predictions in their respective asset classes [2].

High-frequency trading (HFT) utilizes hardware acceleration to reduce delays from microseconds to nanoseconds. The primary focus of the HFT community revolves around executing trades at exceptionally high speeds [8]. The HFT community targets minimum latency to send orders to the market. While the intense competition in speed among HFT firms may have enabled them to achieve significant financial gains [9], the secretive use of novel strategies and techniques to outperform one another has constrained the growth of this dynamic field within the research community [10]. The lack of hardware architectures for prediction algorithms not only hinders the exploration of novel research-based advancements in this field, but also slows down the adoption of new strategies such as hardware/software codesign.

A technical indicator serves as a tool utilized by traders and investors within stock markets to analyze past price movements and additional market data. It identifies patterns and trends within historical data that may predict future price shifts, helping in the decision-making process regarding stock purchases or sales [11]. Through the utilization of technical indicators, traders can obtain insights into market sentiment, momentum, volatility, and potential turning points in stock prices [12,13]. This information proves valuable for determining optimal times to enter or exit trades, managing risk, and maximizing returns within the stock market [14]. Various technical indicators are employed in stock market forecasting. When selecting these indicators, several factors are taken into account, including their popularity, profitability,

* Corresponding author.
*E-mail address:* mhzafar@cardiffmet.ac.uk (M.H. Zafar).

https://doi.org/10.1016/j.aej.2024.03.064
Received 12 August 2023; Received in revised form 31 January 2024; Accepted 20 March 2024
Available online 5 April 2024

suitability for integration into machine learning algorithms, and compatibility with hardware systems [2,15,16].

Technical indicators such as Moving Average Convergence/Divergence (MACD) [2,17,18], Relative Strength Index (RSI) [2,17,19], and Aroon [2,20] are extensively employed in stock market forecasting. This paper presents multiple hardware designs for each of these three technical indicators, employing critical path analysis. During the hardware design process, techniques like pipelining, folding, and circuit replication are utilized to divide the longest path contributing to delays. Despite the widespread utilization of technical indicators for stock market prediction, as far as we are aware, their hardware designs have not been documented in existing literature. Therefore, this paper represents the initial effort to present detailed hardware architectures and conduct a comprehensive analysis for performance optimization.

The main goal of the proposed system is to create a specialized and adaptable framework for hardware architectures developed to execute technical indicators for financial algorithms. With few documented hardware architectures available in existing literature, there is significant room for the development of advanced and efficient hardware designs capable of transmitting precise orders to the market. This research aims to enhance the speed of strategies for stock market prediction in hardware while maintaining the same level of accuracy as their software counterparts. The ultimate objective is to achieve an optimal balance between speed and efficiency.

The rest of the manuscript is organized as follows: A comprehensive literature survey is presented in Section 2, and the major research contributions of this work is listed in Section 3. Section 4 discusses the design and algorithmic choices to achieve a balanced trade-off between efficiency and speed. Section 5 focuses on the hardware implementations of the three selected trading algorithms using the ZYNQ SoC platform. The evolution of the proposed hardware architectures through critical path analysis and their performance evaluation is discussed in Section 6. Lastly, Section 7 concludes the paper, outlining future directions for research.

## 2. Literature survey

There is a huge gap between the literature available for software - based algorithmic trading methods [3] and the HFT related techniques [4]. The main goal of the researchers and developers targeting software-based methods is come up with the strategies to maximize the accuracy of predictions [2]. When examining software-oriented approaches, the literature extensively covers and discusses the most commonly used methods for predicting stock market prices, such as fundamental analysis and technical analysis [2], followed by more advanced techniques like Machine Learning (ML) and sentiment analysis [21]. Software-oriented approaches in stock market prediction have some limitations, including longer latency and lack of scalability. Software implementations can be slow, especially when dealing with large datasets or complex algorithms [22]. This delay impacts real-time decision-making in fast-paced trading environments. As the volume of data increases or the complexity of algorithms grows, software-based systems may struggle to scale efficiently, leading to performance degradation [23]. Hardware approaches can overcome these issues by offering low latency and efficient implementation. Hardware implementations can achieve significantly faster processing speeds compared to software, enabling real-time analysis and decision-making [24]. These architectures can be designed to efficiently handle large volumes of data and complex algorithms, ensuring consistent performance as workload increases [25,30].

The hardware designs documented in literature over the past 12 years are referenced [8] and from [26] to [35]. Recent research contributions in System-on-Chip (SoC) design using Field Programmable Gate Arrays (FPGAs) are highlighted in [36] and [37]. Some earlier publications, such as [26,27], describe attempts to implement a gigabit UDP/IP network stack using FPGA acceleration. Another publication, [29], introduces a hardware architecture for storing market state based on incoming messages from the exchange. [30] showcases a customized architecture for market-data processing with a latency of 307 nanoseconds per message, boasting twice the speed of its software counterpart. Similarly, [31] focuses on implementing only a portion of the entire system. Huang et al. in [34] implemented their design on a Xilinx Alveo U280 acceleration card, utilizing over 8000 DSP units in their DNN-based architecture. While these DSP units have the potential to accelerate individual operations, the cascading of numerous DSP units adds to the routing delays needed to connect the logic to other combinational blocks in the design. Aside from [34] and [35], little discussion is available regarding the implementation of financial algorithms. Both designs adhere to a fixed approach, with limited room for scalability and expansion.

In summary, literature on algorithmic trading using software-based approaches is much more extensive than that on High-Frequency Trading (HFT) [4]. There are very few documented HFT designs in literature [8,28,29,33,35], and those available are either incompletely implemented on reported hardware platforms [8] and [26] to [30] or exhibit suboptimal performance [31,32] to [35]. A technical indicator is a tool used to identify patterns and trends within historical data that can predict future price shifts, aiding in the decision-making process regarding stock purchases or sales. Despite the widespread use of these technical indicators for stock market prediction, to the best of our knowledge, their hardware designs have not been documented in existing literature. Therefore, this paper represents the initial effort to present detailed hardware architectures and conduct a comprehensive analysis for performance optimization.

## 3. Research contribution

Our major contributions are:

- Efficient hardware architectures for three technical indicators (MACD, RSI, and Aroon) optimized for minimal latency through critical path analysis.
- Investigation of combinations of technical indicators capable of operating concurrently in hardware with minimal disparities in throughput, enhancing prediction accuracy.
- Exploration of design space for trading systems to balance speed and efficiency, proposing optimal hardware/software partitioning approaches for complex designs leveraging technical indicators as inputs to machine learning algorithms.

## 4. Design and algorithmic choices for efficiency and speed

This section discusses all the significant design decisions, covering the platform selection, datasets utilized, potential algorithmic options and selections, and various design alternatives in terms of hardware/software partitioning.

**Platform and Optimization Target.** To create a flexible trading platform, the Xilinx Zynq-7000 programmable SoC(XC7Z020-CLG484-1) is selected, featuring an ARM-based microprocessor system in its Processing System (PS) and a 28 nm Artix-7 based reconfigurable logic in its programmable logic (PL). The objective is to minimize design latency for high-speed usage of indicators, aiming for a set of precise and swift indicators to boost overall system efficiency when multiple indicators are employed. Integrating technical indicators and ML algorithms in hybrid models can be challenging due to potential complexities and prolonged development times in complete hardware designs. Thus, adopting a System-on-Chip (SoC) framework proves practical, particularly for facilitating smart hardware/software (HW/SW) partitioning. This strategic approach involves placing compute-intensive tasks in the hardware, enabling low-frequency trading systems to compete effectively with mid-range setups and enhancing prediction efficiency.

**Algorithmic Choices.** According to a systematic review conducted in [2], 66% of the articles reviewed used technical analysis for stock

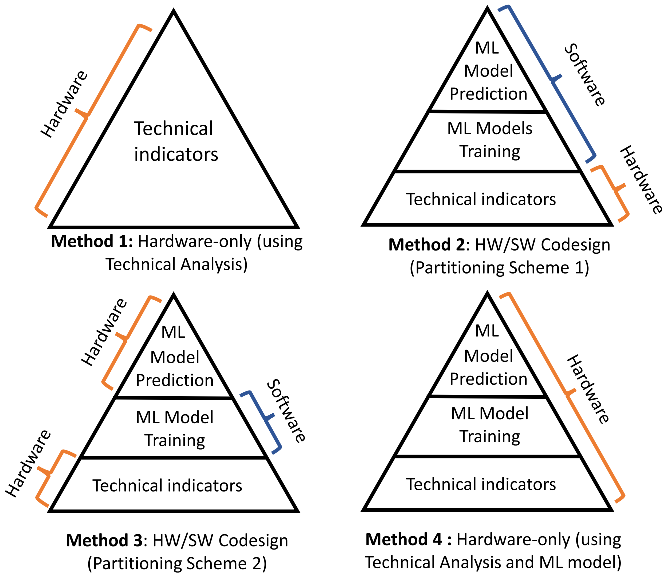


**Fig. 1.** Possible HW/SW partitioning schemes for standalone (purely technical indicator based) and hybrid (technical indicators are used as an input to ML models) trading system.

market prediction. In some cases, a hybrid approach is also followed, where technical indicators are used as an input to the ML models for stock market forecasting [38], [39], [40], [41], [42]. Technical indicators are classified into four main categories: overlap, momentum, volume, and volatility indicators. A comprehensive study in [16] assesses the impact of these categories on enhancing prediction capability in hybrid models. The study reveals that momentum indicators significantly enhance prediction accuracy compared to other groups. Thus, incorporating momentum technical indicators is considered a viable strategy to improve prediction accuracy in both standalone and hybrid models.

At the algorithmic level of abstraction, the three commonly used momentum technical indicators including MACD [16], RSI [19], and Aroon [20] are selected to predict the direction of the stock market.

**Dataset Used.** In this study, the proposed methodology is experimentally evaluated using the two leading cryptocurrencies, Bitcoin (BTC) [43] and Ethereum (ETH) [44]. Data is obtained from Ducascopy-bank [45], spanning from Jan 01, 2021, to Dec 31, 2021. According to [46], Bitcoin price fluctuations are primarily driven by market sentiments and momentum rather than economic variables, making it a suitable choice for the dataset.

**Design Choices.** Four viable design options are proposed for HW/SW partitioning in stock market prediction, as depicted in Fig. 1. These options are further compared and contrasted, taking into consideration various design factors such as latency, prediction efficiency, design complexity, computational complexity, and development time.

In Method 1, the technical indicators are implemented purely in hardware. Since speed is paramount alongside accuracy, a practical approach is to implement all indicators in hardware rather than software. Offloading these designs to hardware enables their data paths to operate in parallel, minimizing the critical path to only the indicator with the longest delay. This significantly speeds up performance compared to their software counterparts, which execute sequentially. While adding indicators in hardware increases reconfigurable resources in terms of area, latency still depends on the design contributing to the critical path.

In Method 2, the predictive capability of ML models is enhanced for stock market prediction by using these indicators as input features to ML models. In this setup, the technical indicators reside on the hardware (PL), with their output serving as input to the software (PS) alongside the ML models, which are also software-based. Method 3 maintains the technical indicators as before, while the ML models are implemented using hardware/software co-design. Method 4 involves a hybrid model entirely implemented in hardware.

**Table 1**
Parametric analysis of all four proposed design choices for stock market prediction. Method 1: Hardware-only (using technical indicators), Method 2: HW/SW co-design (Partitioning scheme 1), Method 3: HW/SW co-design (Partitioning scheme 2), Method 4: Hardware-only (hybrid Model).

| Parameters | Method 1 | Method 2 | Method 3 | Method 4 |
|---|---|---|---|---|
| Accuracy | Medium | High | High | High |
| Latency | Low (A+) | High | Medium | Low (A) |
| Feasibility | High | High | High (A-) | Low |
| Development Time | Low | Medium | High(A-) | High (A+) |
| % Coverage | High | Medium | Medium | Medium |

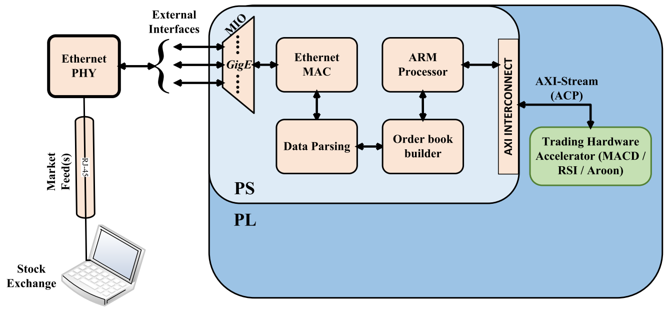


**Fig. 2.** A Framework for High-Frequency Trading Algorithms on FPGA using Zynq SoC Platform.

As shown in Table 1, Method 1 exhibits the lowest latency for stock market predictions, albeit with the lowest accuracy compared to other approaches. Similarly, Method 4 offers lower latency compared to the second and third methods while maintaining similar accuracy levels. However, its development time is the longest among all four approaches.

In the framework utilized in this study, the test bed is configured for all approaches, with Method 1 serving as the foundation for implementing an expandable setup for all other methods. Employing SoC rather than an FPGA card for implementing the designs allows for system scalability by enhancing the feature set in software.

Fig. 2 offers a clear picture of how data smoothly moves from the market to the Zynq board, passing through decoding in the PS, analysis in the Order Book, and algorithmic decision-making in the PL.

## 5. Implementation of algorithms in hardware

This section covers the basic architectures for MACD, RSI, and Aroon, along with their design methodology, forming the foundation for critical path analysis. Their subsequent versions are developed based on the delays caused by the longest paths in these designs for all three algorithms. Optimizations aimed at enhancing performance are discussed in Section 5, along with experimental results.

### 5.1. MACD algorithm

MACD belongs to the family of momentum indicators, employed to identify the trend direction, momentum, and potential reversal points [18]. It consists of two components: the MACD line and the signal line. The MACD line represents the difference between the 12-day Exponential Moving Average (EMA) and the 26-day EMA [47], while the signal line is a 9-day EMA of the MACD line [48], as shown in equations (1) and (2).

$$\mathrm{MACD} = \mathrm{EMA}_{12}(C_p) - \mathrm{EMA}_{26}(C_p). \tag{1}$$

**Table 2**
Smoothing factor "K" for all EMA Modules used in the traditional algorithm, and the corresponding modified smoothing factors "$C_m$" for $EMA_{12}$, $EMA_{26}$, and $EMA_9$, where $1 \leq m \leq 3$.

| Modules | Traditional | | Modified | | |
|---|---|---|---|---|---|
| | Smoothing Factor, $K$ | | Smoothing Factor, $C_m$ | | |
| | $K$ | $1\text{-}K$ | $C_1$ | $C_2$ | $C_3$ |
| $EMA_{12}$ | 2/13 | 11/13 | 5,461 | 55,453 | 10,082 |
| $EMA_{26}$ | 2/27 | 25/27 | 2,520 | 60,681 | 4,854 |
| $EMA_9$ | 1/5 | 4/5 | 7,281 | 52,428 | 13,107 |

$$\text{MACD Signal} = \text{EMA}_9(MACD), \quad (2)$$

where, $C_p$ is the stock closing price of the day and an EMA is a moving average (MA) that gives more importance and emphasis to the latest data points.

MACD also employs a smoothing factor for each of the three moving averages, denoted by $K$ as shown in Table 2, where $K = \frac{2}{N+1}$ and $N = (12, 26, 9)$ [47] and [48]. The smoothing factor K depends on the number of days required to calculate each of the three moving averages.

To generate buy or sell signals for traders, the signal line is plotted over the MACD line. A buy signal is typically indicated by the MACD line crossing above the signal line, interpreted as a bullish signal suggesting the asset's price may rise. Conversely, a sell signal occurs when the MACD line crosses below the signal line, indicating a potential decline in the asset's price.

A hardware implementation of the MACD algorithm using the traditional smoothing factor "K" requires two division operations for each of the three Exponential Moving Averages (EMAs), resulting in a total of six divisions. In the built prototype architecture, six ROMs were instantiated for this purpose. However, the time needed for synthesis and implementation of the design became quickly impractical. To mitigate the implementation time and critical path delay, a modified MACD algorithm is proposed, designed for hardware implementations by utilizing a modified smoothing factor denoted as $C_m$, as indicated in Table 2.

Algorithm 1 shows the entire sequence of operations performed in hardware to generate the decision order. For ease of understanding, the smoothing factors $C_1$, $C_2$ and $C_3$ from Table 2 are denoted with an expanded notation in Algorithm 1 to indicate their affiliation with one of the three EMAs. For example, $C_{12,j}$ refers to the smoothing coefficients required for $EMA_{12}$ where $1 \leq j \leq 3$.

In the modified MACD algorithm, $EMA_{12}$ is computed from lines 3-10. The three constants are utilized in lines 4 and 6 respectively. In the original algorithm, these operations would have required dividers in hardware. To avoid this, a property from digital design was applied to replace the dividers with a constant multiplication and right shift operation.

Consider the operation X/Y, where the divisor Y is not a power of 2. To have a divisor that is also a power of 2, the following steps are performed:

$$\frac{X}{Y} * \frac{2^{16}}{2^{16}} = \frac{X}{2^{16}} * C_m$$

where, $C_m$ is the modified smoothing factor computed as $\frac{2^{16}}{Y}$. In a similar manner, lines 12-20 and lines 23-33 are used to compute $EMA_{26}$ and $EMA_9$ respectively. As shown in Fig. 3, the best accuracy in terms of buy and sell signals was achieved using $2^{16}$ where 20 decision orders were taken as opposed to 14 orders for $2^{15}$.

The design takes 26 cycles to compute $EMA_{26}$, followed by 9 more cycles for the 9-day exponential moving average of MACD which constitutes the first MACD signal's value. The entire algorithm takes 34 cycles to generate the first order signal, i.e., buy or sell (Fig. 4).

The block diagram of a generic EMA module for each of the three variants can be seen in Fig. 5. One input to this block will always be the $Sum_N$, i.e., $Sum_{12}$, $Sum_{26}$ and $Sum_9$. For the second input, $EMA_{12}$ and $EMA_{26}$ utilize the close prices whereas, whereas MACD is employed as

**Algorithm 1** Modified MACD Algorithm.

```
Input: $C_p$ - close prices of all cyrprocurrencies; N - number of close prices;
    $C_{12,j}$, $C_{26,j}$, $C_{9,j}$ - modified smoothing factors for $EMA_{12}$, $EMA_{26}$ and $EMA_9$
    respectively; $1 \leq j \leq 3$
Output: MACD, $EMA_9 \in$ MACD signal
 1: for $i = 0$ to $N$ do
 2:     $Sum = \sum_{n=1}^{26} C_p(i)$
 3:     if $i == 12$ then
 4:         $EMA_{12}(i) = Sum * C_{12,1}$
 5:         $EMA_{12}(i) = EMA_{12}(i) >> 16$
 6:     else if $i > 12$ then
 7:         $EMA_{12}(i) = C_p * C_{12,2} + EMA_{12}(i-1) * C_{12,3}$
 8:         $EMA_{12}(i) = EMA_{12}(i) >> 16$
 9:     else
10:         $EMA_{12}(i) = 0.$
11:     end if
12:     if $i == 26$ then
13:         $EMA_{26}(i) = Sum * C_{26,1}$
14:         $EMA_{26}(i) = EMA_{26}(i) >> 16$
15:     else if $i > 26$ then
16:         $EMA_{26}(i) = C_p * C_{26,2} + EMA_{26}(i-1) * C_{26,3}$
17:         $EMA_{26}(i) = EMA_{26}(i) >> 16$
18:     else
19:         $EMA_{26}(i) = 0.$
20:     end if
21: end for
22: $Sum = \emptyset$
23: for $i = 26$ to $N$ do
24:     MACD $(i) = EMA_{12}(i) - EMA_{26}(i)$.
25:     $Sum = \sum_{n=1}^{9}$ MACD$(i)$.
26:     if $i == 34$ then
27:         $EMA_9(i) = Sum * C_{9,1}$
28:         $EMA_9(i) = EMA_9(i) >> 16$
29:     else if $i > 34$ then
30:         $EMA_9(i) = MACD(i) * C_{9,2} + EMA_9(i-1) * C_{9,3}$
31:         $EMA_9(i) = EMA_9(i) >> 16$
32:     else
33:         $EMA_9(i) = 0.$
34:     end if
35: end for
```

the second input for $EMA_9$. The output of the adder, after being right-shifted by 16, enters a feedback loop until the output is computed.

The data excess pattern of MACD algorithm is illustrated in Fig. 6. The computation of $EMA_{12}$, $EMA_{26}$ and $EMA_9$ requires 12, 26 and 35 cycles respectively. Moreover, the output is generated for the length of the dataset denoted as $'a'$, which necessitates an additional three clock cycles, as indicated by the counter reaching $'a + 3'$ in the data access pattern.

## 5.2. RSI algorithm

The Relative Strength Index (RSI) is a widely used momentum indicator that signals the extent of recent gains and losses in the price of stocks or other asset classes. Mathematically, it can be represented as follows:

$$RSI_i = 100 - \frac{100}{1 + RS_i}, \ (14 \leq i \leq a) \quad (3)$$

where, relative strength (RS) is the ratio of average gain (AG) to average loss (AL).

$$RS_i = \frac{AG_i}{AL_i}, \ (14 \leq i \leq a) \quad (4)$$

The entire sequence of operations is described in Algorithm 2, where $a$ is the length of the file.

The computation begins with an average initial gain ($AG_{14}$) and average initial loss ($AL_{14}$) accumulated over the last 14 days. The average gain and loss shown in Algorithm 2 are computed as follows:

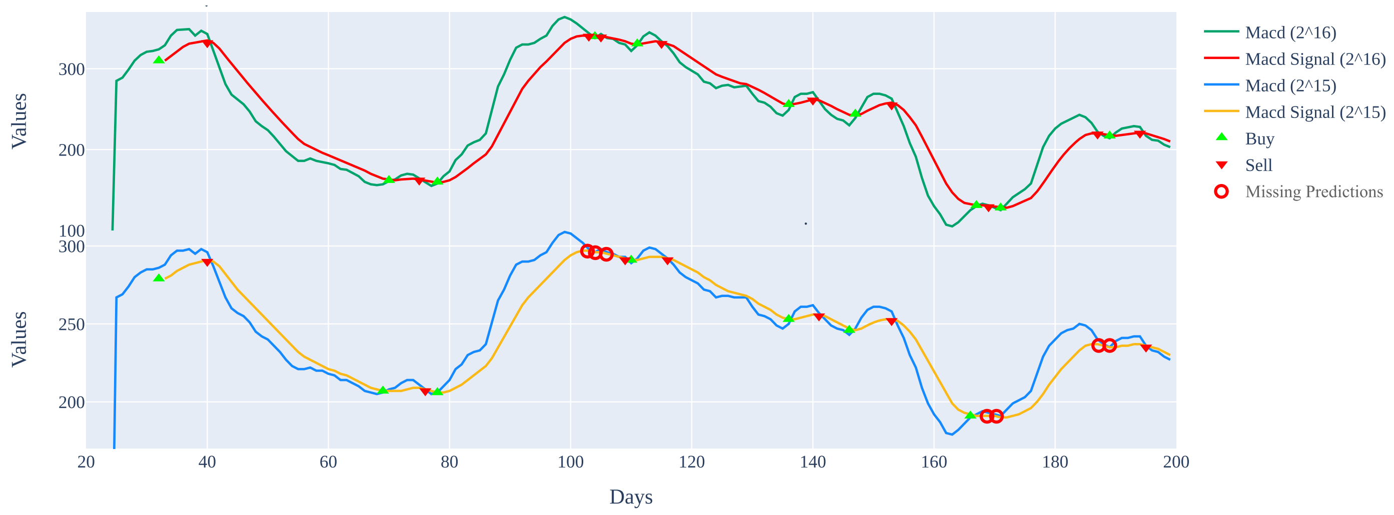


**Fig. 3.** Number of correct MACD predictions using $2^{15}$ and $2^{16}$.

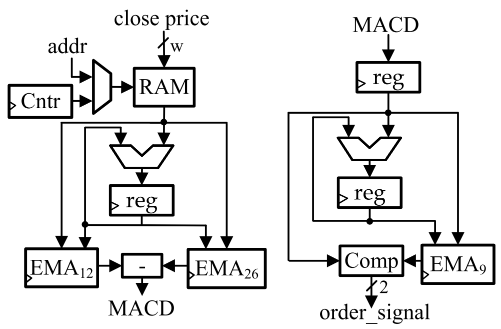


**Fig. 4.** Block diagram of the MACD datapath.

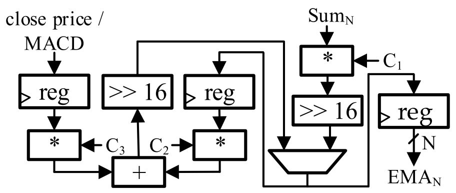


**Fig. 5.** Datapath of the EMA$_N$ Module.

**Algorithm 2** RSI Algorithm.

```
Input: $Cp_i \in Close\ Prices$
       $Cp_i = (Cp_0, Cp_1 ... Cp_a), 0 \le i \le a$
Output: $RSI_j$
 1: for $i = 1$ to $a$ do
 2:     if $Cp_i - Cp_{i-1} > 0$ then $Gain_i = Cp_i - Cp_{i-1}$
 3:     else $Gain_i = 0$
 4:     end if
 5:     if $Cp_i - Cp_{i-1} < 0$ then $Loss_i = |Cp_i - Cp_{i-1}|$
 6:     else $Loss_i = 0$
 7:     end if
 8: end for
 9: for $j = 14$ to $a$ do
10:     if $j = 14$ then $RS_{14} = \frac{AG_{14}}{AL_{14}}$
11:     else $RS_j = \frac{(AG_{j-1}*13+Gain_j)/14}{(AL_{j-1}*13+Loss_j)/14}$
12:     end if
13:      $RSI_j = 100 - \frac{100}{1+RS_j}$
14: end for
```

$$AG_i = \begin{cases} \frac{1}{n}\sum_{i=1}^{14} Gain_i, & \text{if } i = 14 \\ \frac{AG_{i-1}*13+Gain_i}{14}, & \text{if } i > 14 \end{cases}$$

$$AL_i = \begin{cases} \frac{1}{n}\sum_{i=1}^{14} Loss_i, & \text{if } i = 14 \\ \frac{AL_{i-1}*13+Loss_i}{14}, & \text{if } i > 14 \end{cases}$$

where $n$ is 14. The period of 14 days is widely used however, it is not a fixed value. It can be adjusted depending on the trading strategy. If prices decreased over the last 14 days, the RSI will be zero. A value of 100 for the RSI indicates that prices went up over the last 14 days, setting $AL_i = 0$. Increasing the number of periods helps to smooth out the values.

The hardware architecture for the RSI data path is depicted in Fig. 7. The initial computation involves two sub-blocks: the Gain Module and the Loss Module. These modules operate simultaneously, reading the difference between the close prices of two consecutive days stored in one of the block RAMs. After accumulation for up to 14 cycles, necessary to compute the initial RS value, the blocks estimate the relative strength (RS) for the entire length of the input file. For each subsequent day (i.e$i > 14$), $AG$ is calculated by passing the sum of $Gain_j$ and $(13 * AG_{j-1})$ to the ROM (to handle division by 14). A similar procedure is followed for the average loss using the Loss module.

The output from the gain module is left-shifted by 6 bits. The outputs of the gain and loss modules are then fed to the divider to compute RS, generated by the ROM. Finally, an RSI value is compared with the previous value to determine a 2-bit order signal. The data access pattern of our data path for RSI is illustrated in Fig. 8, where $Diff_i$ represents the difference between close prices of consecutive days, $G_i$ denotes the gain, and $L_i$ denotes the loss for the ith day. For a data file of length “a”, the total number of clock cycles required for the entire computation will be “a + 15” cycles due to a register delay (three cycles) and time for division (12 cycles).

A multiplication factor of $2^n$ is utilized to convert a floating-point RS value into a decimal number. This approach is necessary to prevent an RSI of zero for a relative strength number between 0 and 1. Choosing a number that is a power of two is advantageous in digital hardware design because multiplying a number by a power of 2 can be achieved with a simple left shift operation by $n$ positions. The value of n is selected based on the desired number of correct buy/sell decisions.

Fig. 9, compares the selection of $n$ (5 versus 6) based on the number of correct decisions taken by the design.

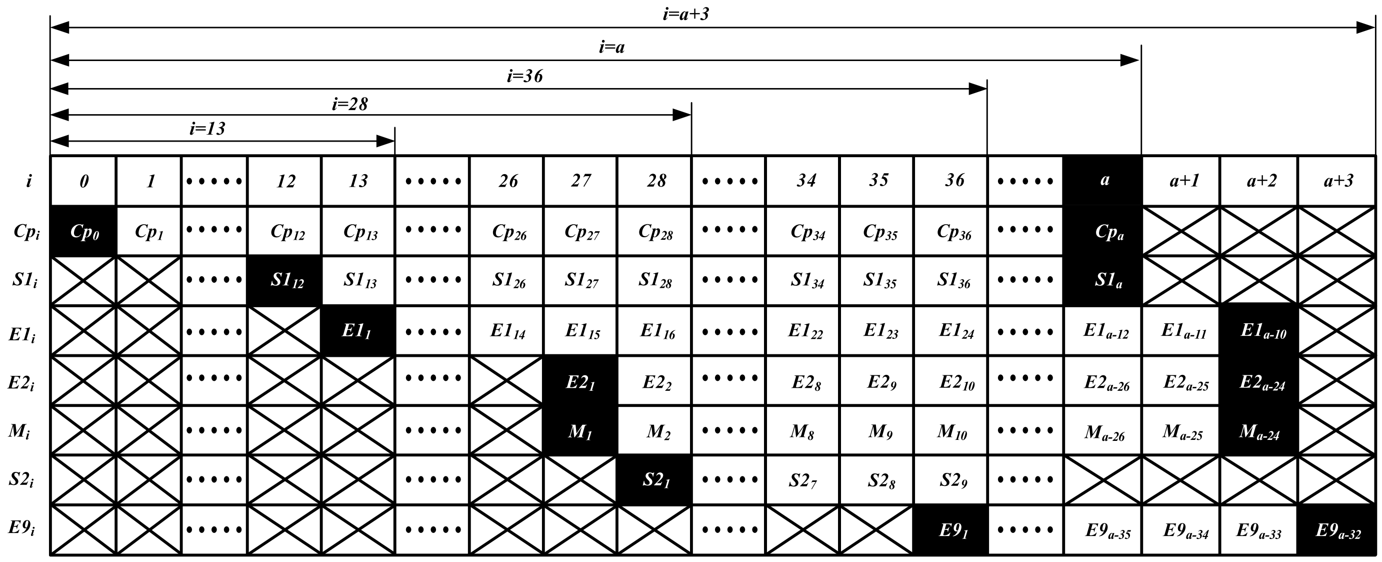


**Fig. 6.** Data excess pattern of MACD. **Notation:** $S1$ - accumulative sum of close prices; $E1$, $E2$ and $E9$ contain values for EMA$_{12}$, EMA$_{26}$ and EMA$_{9}$ respectively; $M$ - MACD; $S2$ - accumulative sum of MACD; $a$ - length of the dataset.

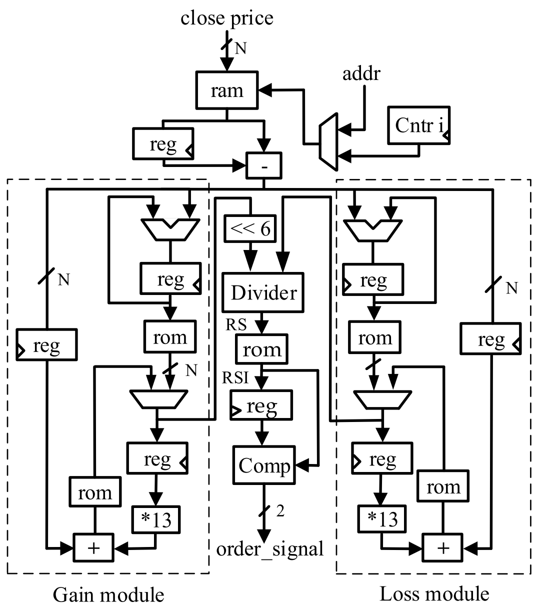


**Fig. 7.** Block diagram of the RSI datapath.

### 5.3. Aroon algorithm

Aroon is a trend-following momentum indicator used to assess the strength of a trend in stock price movement and the likelihood of its continuation. It comprises two lines: Aroon Up and Aroon Down [20], which fluctuate between 0 and 100 [17]. The formulas for these lines are provided below.

$$\text{Aroon}_{up} = \left(\frac{\text{Time Period} - X}{\text{Time Period}}\right) * 100.$$

$$\text{Aroon}_{down} = \left(\frac{\text{Time Period} - Y}{\text{Time Period}}\right) * 100,$$

where, X represents the number of days since the highest high, and Y represents the number of days since the lowest low. The time period typically considered is 14 days [17]. The final formula for the Aroon oscillator is:

$$\text{Aroon Oscillator} = \text{Aroon}_{up} - \text{Aroon}_{down}.$$

When the Aroon Up line exceeds 70, it indicates a strong upward trend and the potential for prices to continue rising. Conversely, if the Aroon Down line surpasses 70, it signals a strong downward trend.

One of the drawbacks of Aroon is, it requires fixed number of clock cycles to generate the decision order, resulting in increased latency compared to both MACD and RSI.

Fig. 10 illustrates the data path of Aroon. It processes both high and low prices simultaneously. The data path starts by calculating the number of occurrences where the price is high/low compared to the previous high/low over the initial period of 14 days.

Algorithm 3 shows the complete sequence of operations of Aroon algorithm.

The data excess pattern of the Aroon algorithm is depicted in Fig. 11. For clarity, “hp” and “lp” represent the high and low prices, respectively. “gt/lw” compares the current and previous high/low prices, while “hc” and “lc” denote the count for the high and low prices during a predetermined time period. Lastly, “Au” and “Ad” represent the values of Aroon Up and Aroon Down, respectively.

## 6. Experimental results

In this section, a thorough analysis of the performance of three momentum indicators on the Xilinx Zynq-7000 programmable SoC XC7Z020-CLG484-1 is presented. All designs are implemented in VHDL and rigorously verified using test benches and test vectors generated using Python scripts. Vivado Design Suite version 2018.2 is utilized for implementation.

Initially, each algorithm is implemented using a basic iterative architecture. These designs undergo optimization by replacing traditionally used components with smarter, time and area-efficient design choices. For instance, in MACD, six dividers are substituted with constant multiplications and right shifts. Subsequently, new variants of all indicators are developed through careful critical path analysis, employing techniques such as pipelining up to N stages.

Each design is discussed individually in the following section, followed by an examination of trade-offs between speed and area, and a ranking of these algorithms based on throughput.

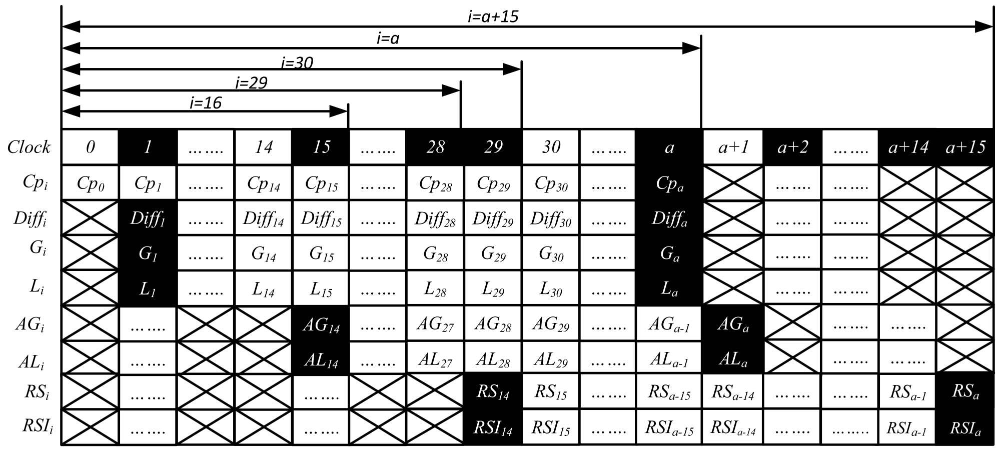


**Fig. 8.** Data Access Pattern of RSI.

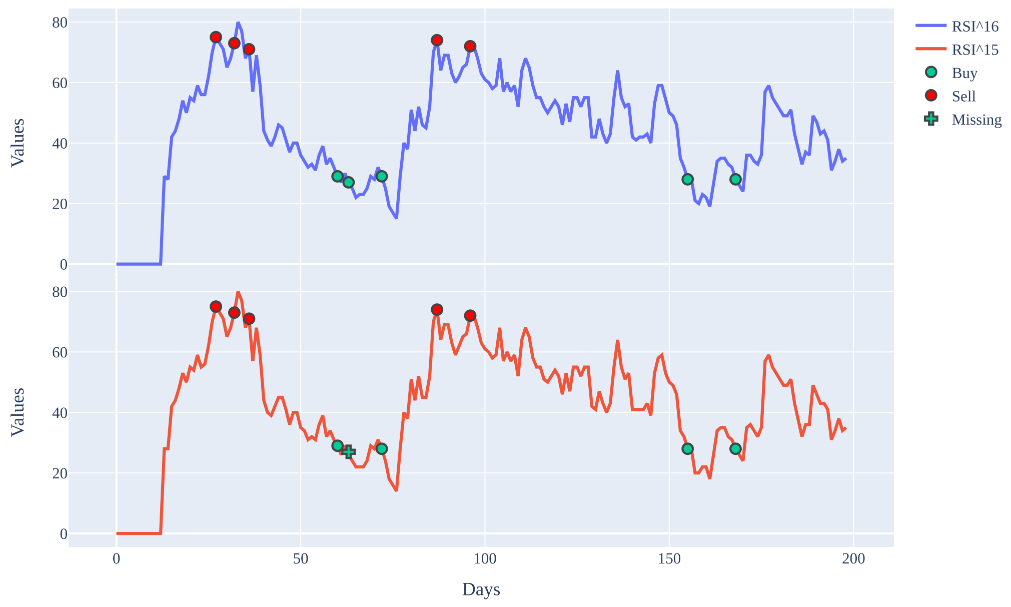


**Fig. 9.** Number of correct RSI predictions using $n = 5$ and $n = 6$.

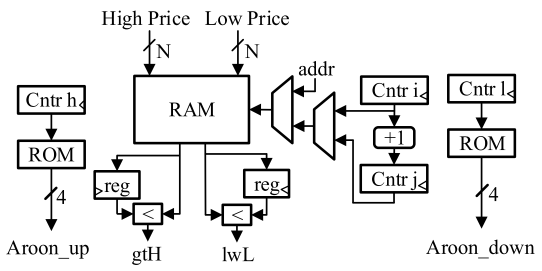


**Fig. 10.** Block diagram of Aroon.

## 6.1. *Evolution of hardware architectures through critical path analysis (CPA)*

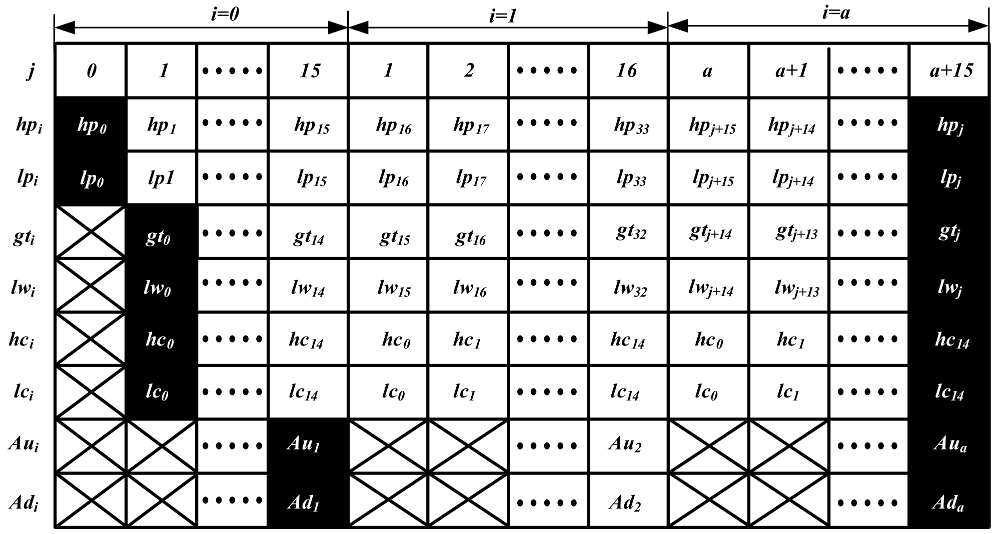
### 6.1.1. *MACD CPA*

In Fig. 12(a), the path highlighted in red represents the critical path of the basic iterative MACD architecture. This path begins at a register within the $\text{EMA}_{12}$ block and extends to a register at the input of another block labeled $\text{EMA}_9$. Along this path, there is a multiplier implemented using DSP units, an adder, a multiplexer, a subtractor, and finally, it terminates at the output of the multiplexer connected to a register input. On the right-hand side, 12(b) illustrates the addition of two registers at the input of a subtractor to alleviate the critical path, resulting in a nearly halved delay.

### 6.1.2. *RSI critical path analysis*

The core operation in the RSI algorithm involves computing the ratio of average gain (AG) to average loss (AL), resulting in the determination of relative strength (RS). Hence, the efficiency of a hardware architecture for RSI depends on discovering the quickest method to execute this operation. With speed as the primary optimization objective, a systematic approach is adopted to minimize the longest path in the design. Initially, the longest path between two registers, which could constrain the design's operational frequency, is identified.

13(a) highlights the critical path of the proposed design, indicated by a red dotted line. The timing report generated by the Vivado IDE indicates positive slack only when the data arrives at the destination before the required time. Originating from the Loss module, the longest path concludes at the input of a register just before the final comparator, which triggers the order signal to the output. Table 3 presents a breakdown of the individual delays caused by all components in the critical path. In the initial design labeled as "RSIbasic" with a combinational



|  | *i=0* |  |  |  | *i=1* |  |  |  | *i=a* |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $j$ | 0 | 1 | ••••• | 15 | 1 | 2 | ••••• | 16 | a | a+1 | ••••• | a+15 |
| $hp_i$ | $hp_0$ | $hp_1$ | ••••• | $hp_{15}$ | $hp_{16}$ | $hp_{17}$ | ••••• | $hp_{33}$ | $hp_{j+15}$ | $hp_{j+14}$ | ••••• | $hp_j$ |
| $lp_i$ | $lp_0$ | $lp1$ | ••••• | $lp_{15}$ | $lp_{16}$ | $lp_{17}$ | ••••• | $lp_{33}$ | $lp_{j+15}$ | $lp_{j+14}$ | ••••• | $lp_j$ |
| $gt_i$ | ☒ | $gt_0$ | ••••• | $gt_{14}$ | $gt_{15}$ | $gt_{16}$ | ••••• | $gt_{32}$ | $gt_{j+14}$ | $gt_{j+13}$ | ••••• | $gt_j$ |
| $lw_i$ | ☒ | $lw_0$ | ••••• | $lw_{14}$ | $lw_{15}$ | $lw_{16}$ | ••••• | $lw_{32}$ | $lw_{j+14}$ | $lw_{j+13}$ | ••••• | $lw_j$ |
| $hc_i$ | ☒ | $hc_0$ | ••••• | $hc_{14}$ | $hc_0$ | $hc_1$ | ••••• | $hc_{14}$ | $hc_0$ | $hc_1$ | ••••• | $hc_{14}$ |
| $lc_i$ | ☒ | $lc_0$ | ••••• | $lc_{14}$ | $lc_0$ | $lc_1$ | ••••• | $lc_{14}$ | $lc_0$ | $lc_1$ | ••••• | $lc_{14}$ |
| $Au_i$ | ☒ | ☒ | ••••• | $Au_1$ | ☒ | ☒ | ••••• | $Au_2$ | ☒ | ☒ | ••••• | $Au_a$ |
| $Ad_i$ | ☒ | ☒ | ••••• | $Ad_1$ | ☒ | ☒ | ••••• | $Ad_2$ | ☒ | ☒ | ••••• | $Ad_a$ |

**Fig. 11.** Data excess pattern of AROON. Notation: $hp, lp$: high and low prices, $gt$: comparison of previous and current high price, $lw$: comparison of the low prices, $hc, lc$: no. of highs and lows during 14 days, $Au, Ad$: $Aroon_{up}$ and $Aroon_{down}$.

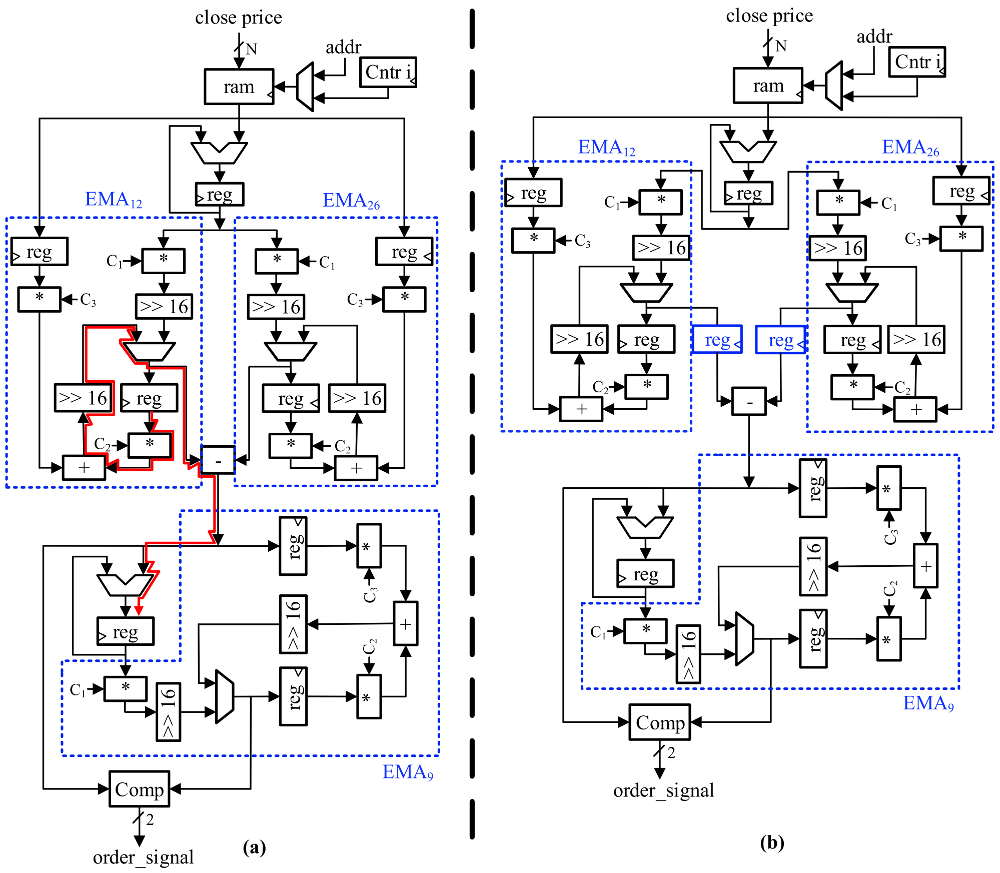


**Fig. 12.** Critical Path Analysis of MACD a) Basic iterative architecture "x1", b) Modified architecture "x2".

divider unit, the total delay generated by each of these components was 56.5 ns.

The second variant of the design involves placing registers at both inputs and the output of the combinational divider. The optimized design, named "RSIppl-d1", is depicted in Fig. 13(b).

To optimize the algorithm, three types of dividers based on the shift/subtract concept are implemented. The first architecture, $d1$, is the basic iterative combinational design, which generates a value in every clock cycle. The second variant, $d2$, features a 2-stage pipelined structure with shift registers. This divider computes a new value after every 12 clock cycles. The third and final architecture, $d3$, boasts a fully pipelined design with N pipeline stages. Initially, it takes 12 clock cycles to fill the pipelined registers. Once the pipeline is filled, a new value can be computed after every clock cycle.

The combinational design $d1$ to compute the ratio $AG/AL$ performs the following operations.

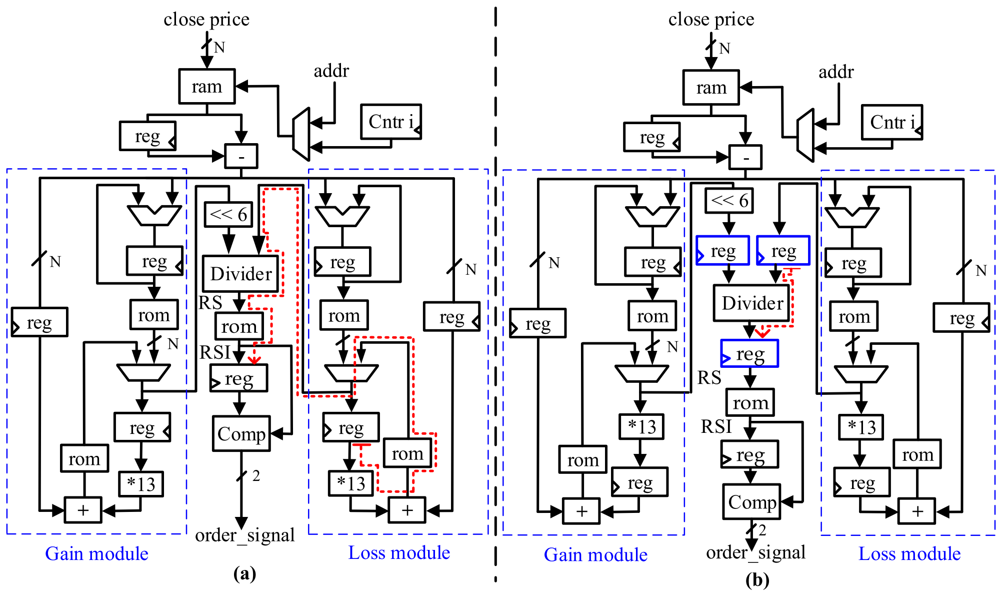


**Fig. 13.** Critical Path Analysis of RSI. a) $\mathrm{RSI}_{\mathrm{basic}}$, b) $\mathrm{RSI}_{\mathrm{ppl\text{-}d1}}$.

**Algorithm 3** Aroon Algorithm.

```
Input: $C_H$: high prices of all cryptocurrencies
       $C_L$: low prices of all cryptocurrencies
       $F_s$: frame size of time period
       $N$: number of required high and low prices
Output: Aroon$_{up}$, Aroon$_{down}$
 1: for $i = 0$ to $N - F_s$ do
 2:     Previous$_{high} = C_H(i)$
 3:     Previous$_{low} = C_L(i)$
 4:     for $j = i + 1$ to $i + F_s$ do
 5:         if $C_H(j) >$ Previous$_{high}$ then
 6:             Previous$_{high} = C_H(j)$, Count$_{up} = \emptyset$
 7:         else
 8:             Count$_{up} =$ Count$_{up} + 1$
 9:         end if
10:         if $C_L(j) <$ Previous$_{low}$ then
11:             Previous$_{low} = C_L(j)$, Count$_{down} = \emptyset$
12:         else
13:             Count$_{down} =$ Count$_{down} + 1$
14:         end if
15:         if $j == i + F_s$ then
16:             Period$_{up} =$ Count$_{up}$
17:             Period$_{down} =$ Count$_{down}$
18:             Aroon$_{up}(j) = \frac{14 - Period_{up}(j)}{14} * 100$
19:             Aroon$_{down}(j) = \frac{14 - Period_{down}(j)}{14} * 100$
20:             Count$_{up} = \emptyset$, Count$_{down} = \emptyset$
21:         end if
22:     end for
23: end for
```

```
begin
d <= CONV_INTEGER(AG);
z <= CONV_INTEGER(AL);
process (z,d)
begin
dividend := z;
divisor  := d;
for i in N downto 0 loop
if (divisor >=(dividend*2**i))then
q(i):= '1';
divisor:= divisor-(dividend*2**i);
else
```

**Table 3**
Detailed summary of the timing delays added by each of the individual components in Fig. 13(a) and (b).

| Path | Start Time (ns) | End Time (ns) | Delay (ns) |
|---|---|---|---|
| Individual component delays in $\mathrm{RSI}_{\mathrm{basic}}$ design | | | |
| Reg → ROM | 5.5 | 10.8 | 5.2 |
| ROM → MUX | 10.8 | 13.0 | 2.1 |
| MUX → Divider | 13.0 | 17.5 | 4.5 |
| Divider → ROM | 17.5 | 56.8 | 39.3 |
| ROM → Reg | 56.8 | 61.6 | 4.7 |
| Individual component delays in $\mathrm{RSI}_{\mathrm{ppl\text{-}d1}}$ design | | | |
| Reg → Divider | 5.0 | 6.4 | 1.4 |
| Divider → Reg | 6.4 | 40.2 | 33.7 |

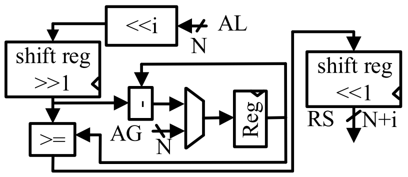


**Fig. 14.** Block Diagram of divider "$d2$", with a 2-stage pipelined structure and shift registers.

```
q(i):= '0';
end if;
end loop;
quot<= q(N downto 0);
end process;
```

Fig. 14 shows the block diagram of divider $d2$. This design, based on shift registers, aims to decrease the critical path delay. It achieves an improvement of over 7 times in terms of delay, with each division requiring 12 clock cycles to compute. The entire dataset spanning a year of values is processed to determine the relative strength. Despite the large number of clock cycles per division, the latency with divider $d1$ is still 1.5 times better than with $d2$, as shown in Table 4.

The block diagram of the third divider, $d3$, is depicted in Fig. 15. In digital design, one approach to maximizing throughput involves un-

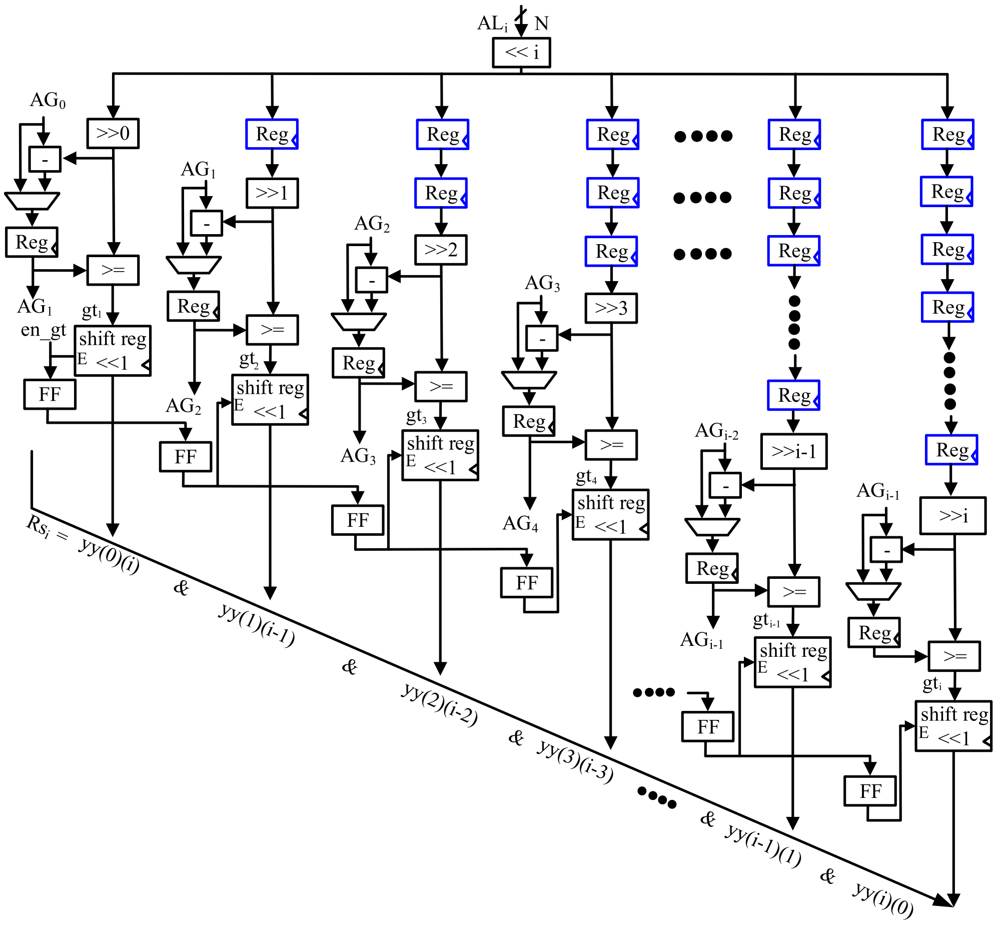


**Fig. 15.** Block Diagram of Fully Pipelined Divider “$d3$”.

**Table 4**
Performance of three divider variants followed by the performance of entire RSI architectures based on these dividers. Notation: d1 - combinational divider; d2 - 2-stage pipelined divider; d3 - Fully pipelined divider.

| Design | Delay (ns) | Freq. (MHz) | Clock Cycles | Latency (ns) |
|---|---|---|---|---|
| d1 | 33.7 | 29.6 | 365 | 12,318 |
| d2 | 4.4 | 227.2 | 4,380 | 19,272 |
| d3 | 5.5 | 181.8 | 377 | 2,074 |
| RSI variants based on the dividers used | | | | |
| $RSI_{ppl\text{-}d1}$ | 35.2 | 28.4 | 368 | 12,957 |
| $RSI_{ppl\text{-}d2}$ | 8.9 | 112 | 4,383 | 39,228 |
| $RSI_{ppl\text{-}d3}$ | 8.5 | 117 | 380 | 3,245 |

rolling and further pipelining the design. By carefully examining the data path structure in design $d2$, a symmetric pattern suitable for logic replication was identified. To enable parallel processing of data, pipelined registers are added at each stage of the design. Initially, it takes 12 clock cycles to fill the pipeline stages. However, once the pipeline is full, a new result is generated in each subsequent clock cycle. The performance of this divider, $d3$, is shown in Table 4. A notable improvement is observed in terms of latency, which is achieved by processing a new input set once the pipeline is filled after 12 cycles initially. The lower half of 4 displays the latency of the entire RSI designs based on all three divider variants. Ultimately, the use of $RSI_{ppl\text{-}d3}$ is recommended as it demonstrates the best performance in terms of both latency and throughput.

### 6.1.3. Aroon critical path analysis

Fig. 16 (a) illustrates the critical path of Aroon indicated by the red line. This critical path extends from a register, through a subtractor and multiplier, and finally terminates at the output of $\text{Aroon}_{down}$. The critical path traverses through a DSP block instantiated in the design. To optimize the design, these computations are executed using a ROM and two counters to minimize the critical path, as depicted in Fig. 16 (b), which introduces an additional delay.

Fig. 17 compares the predictions made by the software implementation with the buy/sell orders generated by the hardware architectures. The plot illustrates that hardware designs provide a significant speedup for all algorithms while maintaining the accuracy achieved by the software versions.

## 6.2. Performance evaluation

The results of all implemented architectures are outlined in Table 5 and Fig. 18. One of the crucial decisions in hardware design is selecting the optimization target, as it influences the entire design process. In this design, the optimization target was speed/throughput. The execution time of a design in terms of the number of clock cycles can be calculated as follows:

$$\text{No. of clock cycles} = \frac{Latency}{T} = \frac{D-L}{T}$$

In the equation above, $D$ represents the time needed to reach the final state, $L$ denotes the start time, and $T$ stands for the time period of the design. The critical path is vital in accelerating an algorithm. The duration of the longest path in the design dictates the frequency of the

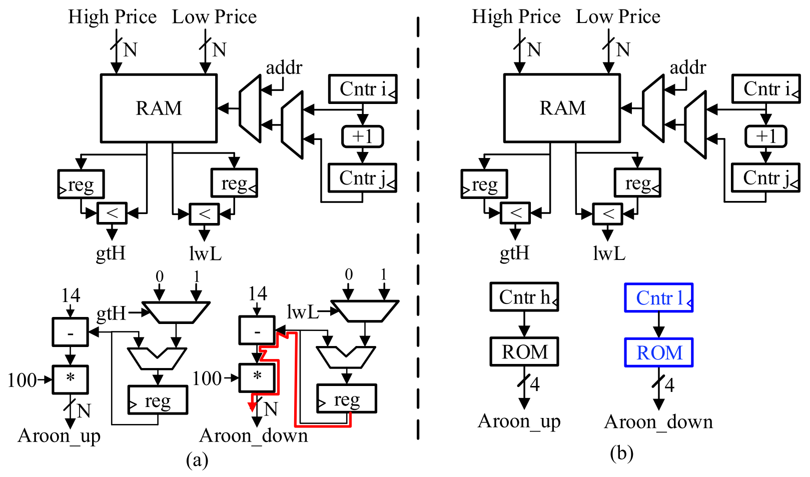


**Fig. 16.** Critical Path Analysis of Aroon. a) x1 b) x2.

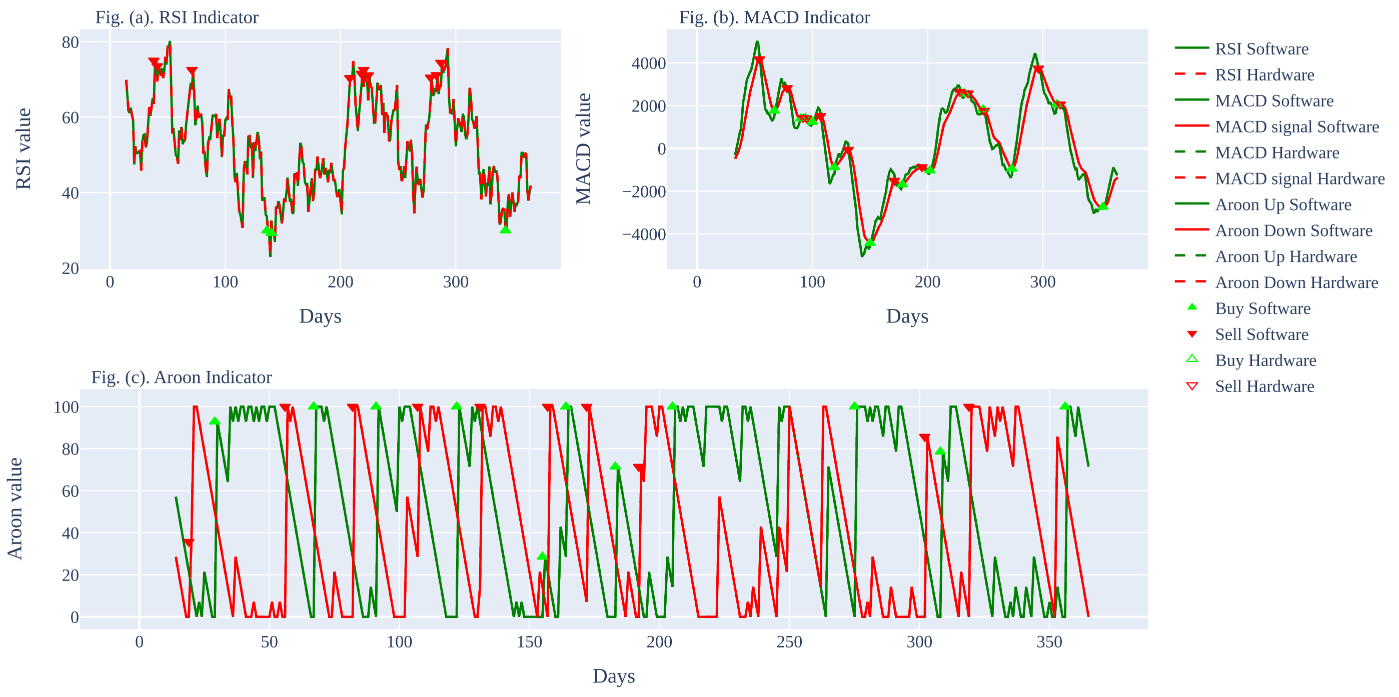


**Fig. 17.** A Comparison of the accuracy achieved by software vs. hardware for a single day time frame on Bitcoin.

**Table 5**
Results for the hardware architectures of all variants of the technical indicators, i.e., MACD, RSI and Aroon.

| Arch. | Clock Cycles | Critical Path (ns) | Freq. (MHz) | Resource Utilization (LUTs, BRAMs, DSPs) | Latency (ns) | Throughput (Mbits/s) |
|---|---|---|---|---|---|---|
| **MACD** | | | | | | |
| x1 | 367 | 14.0 | 71.3 | (419, 0, 9) | 5,143 | 3.9 |
| x2 | 368 | 7.7 | 129.6 | (252, 1, 9) | 2,839 | 7.0 |
| **RSI** | | | | | | |
| ppl-d1 | 368 | 35.2 | 28.4 | (1488, 0, 2) | 12,957 | 1.5 |
| ppl-d2 | 4383 | 8.9 | 112.1 | (1004, 1, 0) | 39,228 | 0.5 |
| ppl-d3 | 377 | 8.5 | 117.0 | (1227, 0, 0) | 3,219 | 6.2 |
| **AROON** | | | | | | |
| x1 | 5616 | 14.3 | 69.8 | (619, 0, 2) | 80,510 | 0.3 |
| x2 | 5616 | 6.5 | 153.1 | (124, 1, 0) | 36,683 | 0.6 |

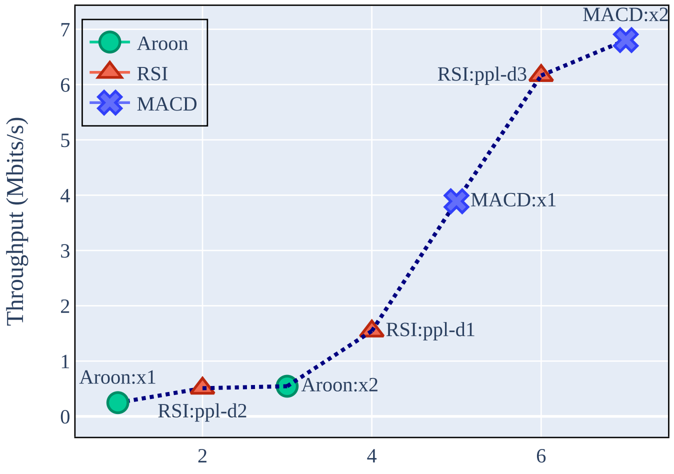


**Fig. 18.** Throughput of implemented hardware architectures.

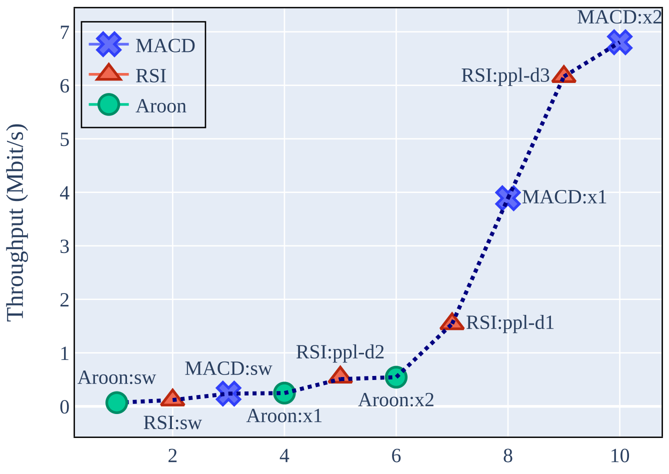


**Fig. 20.** Highest performing software and hardware architectures for all three indicators.

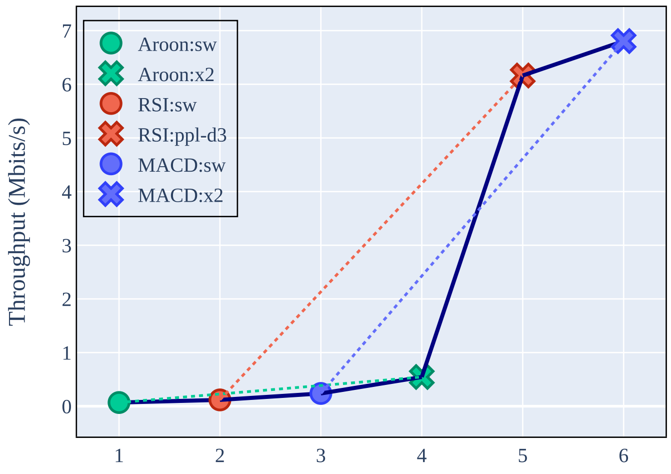


**Fig. 19.** Speedup of hardware designs against their software counterparts.

**Table 6**
Comparison of the best hardware architecture from each of the technical indicators with the software implementation of the respective algorithm.

| Algorithm | Arch. | Latency (ns) | Throughput (Mbits/s) | Speedup |
|---|---|---|---|---|
| MACD | SW<br>x2 | 84,000<br>2,839 | 0.24<br>7.0 | 30* SW |
| RSI | SW<br>ppld3 | 170,000<br>3,219 | 0.1<br>6.2 | 52* SW |
| Aroon | SW<br>x2 | 289,000<br>36,683 | 0.1<br>0.6 | 32* SW |

design. Latency can be enhanced by decreasing the number of clock cycles needed to execute an operation in hardware and/or minimizing the critical path delay.

For MACD, the optimal architecture is “x2”, a variant of the basic iterative design with an additional stage of pipelined registers at the input of the subtractors. This addition reduced the critical path from 14 ns to 7.7 ns, resulting in a 1.8 times improvement in throughput with just one extra clock cycle, as indicated in Table 5. Concerning RSI, the “ppl-d3” architecture achieves the highest throughput without increasing resource utilization. The “x2” architecture for Aroon doubles the throughput compared to the “x1” design, with a 5x reduction in area by simply adding one block RAM to the implementation. As depicted in Fig. 18, among all implemented designs, MACD:x2 exhibits the highest speed, with RSI:ppl-d3 coming in second. Software implementations of all algorithms are slower than the least performing basic hardware design (x1) of Aroon, as shown in Fig. 20.

Table 6 compares the performance of the optimal designs for different trading algorithms in both hardware using ZYNQ SoC and their software implementations. Fig. 19 illustrates a continuous line representing the increasing trend of throughput among multiple variants implemented during the design process. In contrast, a dotted line depicts the performance of software compared to their hardware counterparts. The speedups achieved in hardware versus software are 30, 52, and 32 times for MACD, RSI, and Aroon, respectively.

## 7. Conclusion

Stock market forecasting is one of the most trending problems currently faced by the financial world. For successful trading automation, the entire process can be implemented in software or hardware leveraging advanced technologies. A technical indicator serves as a tool utilized by traders and investors within stock markets to analyze past price movements and additional market data. It identifies patterns and trends within historical data that may predict future price shifts.

Software-based technical indicators have been widely used for the stock market forecasting, aiming to predict market direction. In this paper, the hardware implementation is presented for three commonly used technical indicators: Moving Average Convergence/Divergence (MACD), Relative Strength Index (RSI), and Aroon. Latency evaluation is conducted for Bitcoin and Ethereum within a single-day timeframe.

To validate the suggested framework, the system was implemented on a Xilinx Zynq-7000 programmable SoC XC7Z020- CLG484-1. The three momentum indicators are implemented on the FPGA in both their basic and optimized forms. The optimized architecture (x2) for MACD has a latency of 2.84 us, which is almost half of the original architecture. The throughput has also increased by a factor of two. The optimized architecture (ppl-d3) for RSI reduces latency by a factor of 4 to 3.22 us while increasing throughput by a factor of 4. In the optimized version of Aroon (x2), latency has been cut in half and throughput has increased by a factor of two. Overall, the optimized version of MACD offers the least latency of 2.84 us and highest throughput of 7 Mbits/s. Since, the latency in the optimized MACD and RSI are almost similar, they are ideal candidates for hybrid designs where more than one indicator is used.

A comparison of the performance between the best hardware and software designs for these three momentum indicators reveals significant advantages in terms of low latency and improved throughput in the hardware versions. Optimized versions of MACD and Aroon in hardware demonstrate a speedup of up to 30 times, while RSI shows an impressive improvement of up to 52 times.

These indicators have been widely used in trading software for over 20 years, and when integrated with machine learning algorithms, they can enhance decision-making in trading environments, benefiting the low-frequency algorithmic trading community.

In future work, the platform could expand to include more hardware architectures for additional technical indicators, validating their potential to improve prediction accuracy. Method 2 could be broadened by integrating over 10 technical indicators with machine learning algorithms to enhance predictive intelligence. Offloading machine learning algorithm testing to hardware could further speed up enhancements. Ultimately, the aim of the proposed platform and strategy is to empower researchers to develop superior designs for algorithmic trading setups across all frequencies.

## Declaration of competing interest

The authors declare that they have no conflicts of interest to report regarding the present study.

## References

[1] D. Shah, H. Isah, F. Zulkernine, Stock market analysis: a review and taxonomy of prediction techniques, Int. J. Financ. Stud. 7 (May 2019), https://doi.org/10.3390/ijfs7020026.

[2] I.K. Nti, A.F. Adekoya, B.A. Weyori, A systematic review of fundamental and technical analysis of stock market predictions, Artif. Intell. Rev. 53 (4) (2020) 3007–3057, https://doi.org/10.1007/s10462-019-09754-z.

[3] F. Rundo, F. Trenta, A.L. di Stallo, S. Battiato, Machine learning for quantitative finance applications: a survey, Appl. Sci. 9 (24) (Dec. 2019) 5574, https://doi.org/10.3390/app9245574.

[4] B. Huang, Y. Huan, L.D. Xu, L. Zheng, Z. Zou, Automated trading systems statistical and machine learning methods and hardware implementation: a survey, Enterp. Inf. Syst. 13 (2019) 132–144, https://doi.org/10.1080/17517575.2018.1493145.

[5] Mirwais Waisi, Advantages and Disadvantages of AI-based Trading and Investing Versus Traditional Methods, 2020.

[6] Z. Hu, Y. Zhao, M. Khushi, A survey of Forex and stock price prediction using deep learning, Appl. Syst. Innov. 4 (1) (Feb. 2021) 9, https://doi.org/10.3390/asi4010009.

[7] M. Baron, J. Brogaard, B. Hagströmer, A. Kirilenko, Risk and return in high-frequency trading, J. Financ. Quant. Anal. 54 (3) (2017) 993–1024, https://doi.org/10.2139/ssrn.2433118.

[8] C. Leber, B. Geib, H. Litz, High frequency trading acceleration using FPGAs, in: 21st International Conference on Field Programmable Logic and Applications, 2011, pp. 317–322.

[9] Daniel Ladley, The high frequency trade off between speed and sophistication, J. Econ. Dyn. Control 116 (2020), https://doi.org/10.1016/j.jedc.2020.103912.

[10] Ekkehart Boehmer, Dan Li, Gideon Saar, The competitive landscape of high-frequency trading firms, Rev. Financ. Stud. 31 (6) (June 2018) 2227–2276, https://doi.org/10.1093/rfs/hhx144.

[11] E. Ahmadi, M. Jasemi, L. Monplaisir, M.A. Nabavi, A. Mahmoodi, P.A. Jam, New efficient hybrid candlestick technical analysis model for stock market timing on the basis of the support vector machine and heuristic algorithms of imperialist competition and genetic, Expert Syst. Appl. 94 (Mar. 2018) 21–31, https://doi.org/10.1016/j.eswa.2017.10.023.

[12] C.L. Osler, Currency orders and exchange rate dynamics: an explanation for the predictive success of technical analysis, J. Finance 58 (5) (Oct. 2003) 1791–1819, https://doi.org/10.1111/1540-6261.00588.

[13] Y. Zhu, G. Zhou, Technical analysis: an asset allocation perspective on the use of moving averages, J. Financ. Econ. 92 (3) (2009) 519–544, https://doi.org/10.1016/j.jfineco.2008.07.002.

[14] S. Bader, F.W. Alhashela, J. Almudhafa, H. Andrew, Can technical analysis generate superior returns in securitized property markets? Evidence from East Asia markets, Pac.-Basin Finance J. 47 (1) (2018) 92–108, https://doi.org/10.1016/j.pacfin.2017.12.005.

[15] Terence Tai-Leung Chong, Wing-Kam Ng, Technical analysis and the London stock exchange: testing the MACD and RSI rules using the FT30, Appl. Econ. Lett. 15 (14) (2008) 1111–1114, https://doi.org/10.1080/13504850600993598.

[16] Y. Lin, S. Liu, H. Yang, H. Wu, Stock trend prediction using candlestick charting and ensemble machine learning techniques with a novelty feature engineering scheme, IEEE Access 9 (2021) 101433–101446, https://doi.org/10.1109/ACCESS.2021.3096825.

[17] Steven Gold, The viability of six popular technical analysis trading rules in determining effective buy and sell signals: MACD, AROON, RSI, SO, OBV, and ADL, J. Appl. Financ. Res. 2 (2015).

[18] Parthkumar Kanani, et al., A ratiocinative concept of algorithmic trading using MACD indicator, in: IEEE 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, Dec. 2021.

[19] Mingyuan Wu, et al., Technical analysis of three stock oscillators testing MACD, RSI and KDJ rules in SH and SZ stock markets, in: IEEE 4th International Conference on Computer Science and Network Technology (ICCSNT), Harbin, China, Dec. 2015.

[20] Jiaqi Pan, et al., The impact of data normalization on stock market prediction: using SVM and technical indicators, Int. Conf. Soft Comput. Data Sci. 652 (Sept. 2016) 72–88, https://doi.org/10.1007/978-981-10-2777-2_7.

[21] C. Lamon, E. Nielsen, E. Redondo, Cryptocurrency Price Prediction Using News and Social Media Sentiment, 2017.

[22] Ohbyung Kwon, Jae Mun Sim, Effects of data set features on the performances of classification algorithms, Expert Syst. Appl. (ISSN 0957-4174) 40 (5) (2013) 1847–1857, https://doi.org/10.1016/j.eswa.2012.09.017.

[23] C.L. Philip Chen, Chun-Yang Zhang, Data-intensive applications, challenges, techniques and technologies: a survey on big data, Inf. Sci. (ISSN 0020-0255) 275 (2014) 314–347, https://doi.org/10.1016/j.ins.2014.01.015.

[24] Zhigao Zheng, Ping Wang, Jing Liu, Shengli Sun, Real-time big data processing framework: challenges and solutions, Appl. Math. Inf. Sci. 9 (6) (2015) 3169.

[25] Kaan Kara, Jana Giceva, Gustavo Alonso, Fpga-based data partitioning, in: Proceedings of the 2017 ACM International Conference on Management of Data, 2017, pp. 433–445.

[26] F.L. Herrmann, G. Perin, J.P.J. de Freitas, R. Bertagnolli, J.B. dos Santos Martins, A Gigabit UDP/IP network stack in FPGA, in: 2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009), 2009, pp. 836–839.

[27] G.W. Morris, D.B. Thomas, W. Luk, FPGA accelerated low-latency market data feed processing, in: 2009 17th IEEE Symposium on High Performance Interconnects, 2009, pp. 83–89.

[28] J.W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, K. Vissers, A low-latency library in FPGA hardware for high-frequency trading (HFT), in: 2012 IEEE 20th Annual Symposium on High-Performance Interconnects, 2012, pp. 9–16.

[29] M. Dvořák, J. Kořenek, Low latency book handling in FPGA for high frequency trading, in: 17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, 2014, pp. 175–178.

[30] Q. Tang, L. Jiang, M. Su, Q. Dai, A pipelined market data processing architecture to overcome financial data dependency, in: 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC), 2016, pp. 1–8.

[31] Qiu Tang, Majing Su, Lei Jiang, Jiajia Yang, Xu Bai, A scalable architecture for low-latency market-data processing on FPGA, in: 2016 IEEE Symposium on Computers and Communication (ISCC), 2016, pp. 597–603.

[32] H. Fu, C. He, W. Luk, W. Li, G. Yang, A nanosecond–level hybrid table design for financial market data generators, in: 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2017, pp. 227–234.

[33] A. Boutros, B. Grady, M. Abbas, P. Chow, Build fast, trade fast: FPGA-based high-frequency trading using high-level synthesis, in: 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2017, pp. 1–6.

[34] B. Huang, et al., AIOC: an all-in-one-card hardware design for financial market trading system, IEEE Trans. Circuits Syst. II, Express Briefs 69 (9) (Sept. 2022) 3894–3898, https://doi.org/10.1109/TCSII.2022.3167312.

[35] Y.-C. Kao, H.-A. Chen, H.-P. Ma, An FPGA-based high-frequency trading system for 10 Gigabit Ethernet with a latency of 433 ns, in: 2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), 2022, pp. 1–4.

[36] S. Das, A.K. Sunaniya, R. Maity, N.P. Maity, Efficient FPGA implementation of corrected reversible contrast mapping algorithm for video watermarking, Microprocess. Microsyst. (2020) 103092, https://doi.org/10.1016/j.micpro.2020.103092.

[37] S. Das, A.K. Sunaniya, R. Maity, N.P. Maity, Efficient FPGA implementation and verification of difference expansion based reversible watermarking with improved time and resource utilization, Microprocess. Microsyst. (2021) 103732, https://doi.org/10.1016/j.micpro.2020.103732.

[38] D. Kumar, S.S. Meghwani, M. Thakur, Proximal support vector machine based hybrid prediction models for trend forecasting in financial markets, J. Comput. Sci. 17 (Nov. 2016) 1–13, https://doi.org/10.1016/j.jocs.2016.07.006.

[39] B. Weng, L. Lu, X. Wang, F.M. Megahed, W. Martinez, Predicting short-term stock prices using ensemble methods and online data sources, Expert Syst. Appl. 112 (Dec. 2018) 258–273, https://doi.org/10.1016/j.eswa.2018.06.016.

[40] J. Patel, S. Shah, P. Thakkar, K. Kotecha, Predicting stock market index using fusion of machine learning techniques, Expert Syst. Appl. 42 (4) (Mar. 2015) 2162–2172, https://doi.org/10.1016/j.eswa.2014.10.031.

[41] F. Zhou, Q. Zhang, D. Sornette, L. Jiang, Cascading logistic regression onto gradient boosted decision trees for forecasting and trading stock indices, Appl. Soft Comput. 84 (Nov. 2019) 105747, https://doi.org/10.1016/j.asoc.2019.105747.

[42] W. Bao, J. Yue, Y. Rao, A deep learning framework for financial time series using stacked autoencoders and long-short term memory, PLoS ONE 12 (7) (Jul. 2017) e0180944, https://doi.org/10.1371/journal.pone.0180944.

[43] Bitcoin, [Online], Available: https://www.bitcoin.com. (Accessed 7 May 2022), 2022.

[44] Ethereum, [Online], Available: https://ethereum.org. (Accessed 7 May 2022), 2022.
[45] Dukascopy, [Online], Available: https://www.dukascopy.com. (Accessed 1 July 2022), 2022.
[46] E.-T. Cheah, J. Fry, Speculative bubbles in Bitcoin markets? An empirical investigation into the fundamental value of Bitcoin, Econ. Lett. 130 (C) (2015) 32–36, https://doi.org/10.1016/j.econlet.2015.02.029.
[47] John J. Murphy, Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications, Penguin, 1999.
[48] Jian Wang, Junseok Kim, Predicting stock price trend using MACD optimized by historical volatility, Math. Probl. Eng. 2018 (2018) 1–12, https://doi.org/10.1155/2018/9280590.