Journal of Signal Processing Systems (2023) 95:559–567
https://doi.org/10.1007/s11265-023-01850-2

A FAST Hardware Decoder Optimized for Template Features to Obtain Order Book Data in Low Latency

Caio C. S. Oliveira[1] · **Vanderlei Bonato**[1]

Received: 23 January 2022 / Revised: 21 October 2022 / Accepted: 29 January 2023 / Published online: 6 May 2023

Abstract
High-Frequency Trading (HFT) systems require high computational performance for real-time trading and data analysis. FAST protocol, an extension of FIX protocol, is one of the main communication pattern adopted by these systems. This work presents an open source hardware component, implemented in Field-Programmable Gate Array (FPGA), to decode market data messages to produce the necessary commands to construct order books in low latency for the *Brasil Bolsa Balcão* B3 stock exchange. The proposed hardware component optimized for a B3 template is able to decode messages at average latency of 0.72us and throughput of 1.4M FAST messages per second. The results are from logs of real messages with average size of 85 bytes each.

Keywords FPGA · FIX/FAST Decoder · HFT

1 Introduction

Along with the advance in communication technologies, the financial market has become highly automated, allowing online access to most of its services, including trading and market data. Algorithms deployed as trading robots can access real time data about offers, trades, and news, and use them in their trading strategies to generate orders to the market in a completely automated mode [1]. This level of integration allows the implementation of High Frequency Trading (HFT) systems, in which the performance of the algorithms are highly dependent on low latency solutions for both market data analysis and trading strategies [2, 3].

Communication with stock exchange systems are mostly available through the Financial Information eXchange (FIX) protocol, available in canonical or in encoded versions, such as FAST (Fix Adapted for STreaming) and SBE (Simple Binary Encoding) [4, 5]. In the canonical format, the messages to be transmitted are excessively verbose and not appropriated for HFT systems. On the other hand, the encoded format reduces the message size significantly, however it needs an extra encoding and decoding step, what increases the computing complexity of the system.

Hardware solutions for the market data decoding have been exploited as a way to improve both throughput and latency [6]. Additionally, as the decoder algorithms are embedded in hardware, the time for decoding each element of the message is known in advance, given to the traders a better prediction about the expected delay for market data access. In this paper, we propose a hardware component based on FPGA that implements a FAST message decoder to access order book data using a B3 template. The main contributions of this paper are as follows:

- a low latency solution optimized for particular template features that can be adopted as an optimized hardware component to construct HFT systems;
- an analysis about data dependency of the FAST decoding engine and its impact on the overall hardware performance;
- a solution that is provided as an open source hardware in Register Transfer Level (RTL) along with a golden model in software that can be expanded to support new templates for any data providers.

This paper is organized as follows: Section 2 introduces a brief review of FIX and FAST protocols, Section 3 presents some related works, Section 4 depicts the proposed hardware

✉ Caio C. S. Oliveira
caiossoliveira@gmail.com

Vanderlei Bonato
vbonato@usp.com

[1] Institute of Mathematical and Computing Sciences, The University of São Paulo, São Paulo, Brazil

implementation, Section 5 shows the experimental results, and Section 7 concludes the paper.

2 FIX/FAST message decoding

FIX messages are ASCII strings composed of fields containing "tag = value" separated by the Start Of Header (SOH) character [7]. For example, the FIX canonical message below (Box 1) represents an order book update command, and will be used from hereon as an example to explain the particularities of the protocol. The SOH delimiters are replaced by vertical bars for viewing.

$$1128{=}9 \mid 35 = X \mid 34 = 732046 \mid 52 = 20130710124510273 \mid 75 = 20130710 \mid 268 = 1 \mid 279 = 0 \mid 269 = 1 \mid 22 = 8 \mid 207 = BVMF \mid 48 = 3940368 \mid 83 = 793 \mid 270 = 2 \mid 273 = 124510269 \mid 271 = 15 \mid 272 = 20130710 \mid 37016 = 20130710 \mid 37017 = 124510269 \mid 37 = 80142490094 \mid 289 = 98 \mid 290 = 5 \mid$$

Box 1: Example of a FIX message.

This set of fields represents the whole message, where the field 35 informs the type of the message, which is a market data message for incremental refreshing, and the fields 268 = 1 (NoMDEntries = 1), 279 = 0 (MDUpdateAction = New), 269 = 1 (MDEntryType = Offer), 48 = 3940368 (SecurityID = 3940368), 270 = 2 (MDEntryPx = 2), 271 = 15 (MDEntrySize = 15), and 290 = 5 (MDEntryPositionNo = 5) are the message payload to update an order book. This message contains instructions to insert a new line at position 5 of the offer side of the order book belonging to the respective Security ID, adding volume 15 at price BRL 2.00. The previous field names are abbreviations of market data information, e.g., MDEntryPx means Market Data Entry Price, which is a field that holds the value of some asset. See [7] for more information about the other tags.

FAST encoding optimizes the representation of this message data, reducing redundancy by encoding the text format "tag = value" into a binary format. The example below (Box 2) shows the previous FIX message encoded as FAST, where a reduction from 240 bytes to 90 bytes was possible. The rectangle inside indicates the unencoded technical header of the message.

00 0B 2B 8E 00 01 00 01 00 50 C0 01 91 2C 57 8E 23 61
18 63 10 30 78 C1 09 4C 57 97 81 7B 3C 09 C0 80 B1 01
70 40 90 06 99 81 82 80 80 3B 2F 40 BD 90 09 4C 57 97 09
4C 57 97 3B 2F 40 BE 80 80 80 80 38 30 31 34 32 34 39 30
30 39 B4 39 B8 86 80 80 80 80 80 80 80 80 80 80

Box 2: Example of a FAST message.

Table 1 B3 FAST message format.

Unencoded header			
MsgSeqNum	NoChunks	CurrentChunk	MsgLength
4 bytes	2 bytes	2 bytes	2 bytes
FAST encoded message			
MsgLength bytes			

Table 1 demonstrates the FAST message anatomy specific for the B3. The first 10 bytes are the unencoded technical header, composed by the MsgSeqNum, the same as tag 34, by the NoChunks and CurrentChunk tags, used when a message exceeds the User Datagram Protocol (UDP) packet limit, and the MsgLength field, which informs the message size in bytes.

The following items introduce the features used for data compression, such as the rules to identify the values of the fields, the present fields in the message, and any other required computations.

2.1 Stop Bit

After reading the technical header fields, a FAST decoder, also known as engine, starts to decode the message body. Since the FAST message fields have no fixed length, to split the sequence of bytes into their respective fields, the protocol defines that the MSB (Most Significant Bit) of each byte represents the stop bit. So, when the MSB of a byte is 1, it means that this byte is the last one of the respective field. Taking the starting three bytes of the previous message (C0 01 91) converted to binary, the two fields are obtained: field 1: 1̲1000000, and field 2: 00000001 1̲0010001. Then, after removing the MSB bit of each byte and concatenating the remaining parts, the two fields result in 40 and 00 91 in hex.

2.2 Presence Map

Each FAST encoded message starts with the Presence Map (PMap), a mask that specifies which fields or sequences are present. The meaning of the existent fields are known by the decoder from the FIX template, which are associated to the respective order of PMap bits. Bit one means the field is present and zero is not. From the previous message example, the byte C0 is the PMap field, resulting in hex 40 (1000000) after removing the MSB bit. The binary number 1000000 means that the first field of the template is present in the stream.

2.3 Template

The FAST messages are decoded based on a static and predefined template that specifies all the fields' locations and contents. Each message type has a unique template id to be correctly decoded. The templates are stored in an XML file shared between the sender and the receiver of the message.

Each template holds a set of rules used by the engine to guide the necessary computation required to decode the message by specifying to which field each value refers to, including their operators, sizes, and meanings. Each field has a name and a type that can be a sequence type (item 2.7) or a primitive type.

2.4 Operators

Each value in the stream can either have or not a respective operation specified in the template. There are some kinds of operators such as copy, increment, delta, default, constant, tailor, and none (no operator defined). Each one has a set of rules to decode the value received from the stream. For example, for a copy operation, the rules specify that the value of the field can be either present in the stream or not. If the PMap bit is set, the actual value is the encoded value present in the stream. When it is not present, the actual value of the field could be the previous value, the initial value set in the template, or absent, depending on the previous status, which can be assigned, undefined, or empty. Moreover, the other operations rules are specified in the FAST specifications manual [8].

2.5 Nullability

The nullability property is applied when a field is optional and has no field operator, which means that there is a kind of representation of NULL value given by the hexadecimal 80. Furthermore, the stream values from a nullable field need to be decremented by 1 during the process if it is a non-negative integer field.

2.6 Decimal numbers

The exponent and the mantissa are received in different fields in the stream, and there is a set of rules to decode them. The mantissa is present in the stream only if the exponent is present in the stream or in the initial value from the template. If these conditions are true, the next set of bytes represents the mantissa field.

2.7 Sequence

A sequence is used as a message loop to transmit different values for a specific set of fields, without the need to transmit the whole message again for each loop iteration. The number of interactions is defined by a field that is located before the sequence definition, the *NoMDEntries* tag. According to the example below, the sequence is used to process a series of market data information, with order book update data, as the MDEntryType and MDEntryPx tags.

```
<sequence name="MDEntries">
    <length name="NoMDEntries" id="268" />
    <string name="MDEntryType" id="269">
        <default value="0" />
    </string>
    <decimal name="MDEntryPx" id="270"
                        presence="optional">
        <copy />
    </decimal>
    ...
</sequence>
```

3 Related Works

Most of the works focus on accelerating one or more modules of the HFT system. Several of these implementations are in software, some in heterogeneous computing with specific hardware along with general-purpose processor, and others as a system-on-chip.

[6] proposes a commercial IP for market data decoding, including other components for network communication, external memory interface and software components for the host-hardware communication. Its hardware runs at 156MHz, achieving latency of 200ns for message parsing. The throughput of the system is 6.1M FIX messages per second, with the FIX messages having an average size of 150 bytes, and the OUCH messages between 12 and 82 bytes length. This work does not use FAST messages.

[9] The work presents a domain-specific accelerator for market data processing, including a FAST decoder, a network component, and a host interface to send the market information to a host server. The system was implemented in a Xilinx Alveo U200 acceleration card. Its hardware runs at 300Mhz, achieving a latency of 0.18us to 0.36 to decode each message. The FAST messages have a size from 18 to 45 bytes of length, with a template varying from 9 to 21 fields.

[10] provides an IP library for the FAST decoder implemented in High Level Synthesis (HLS). The decoders were implemented in a Xilinx Kintex-7 FPGA using templates with 9, 11, and 21 fields. Depending on these sizes, the frequency varied from 200MHz to 155MHz. The hardware reached an average of 0.5us to 1.3us to decode each message, achieving speed-up up to 2 orders of magnitude than an equivalent software solution.

[3] also presents an HLS-based system that implements hardware components for a complete HFT system. The system runs at 156MHz, with 270ns average of round-trip latency, without considering the network stack, and 115ns of latency for the FAST decoder specific part. The work creates

its own FAST template to better use the FPGA resources by computing only the needed fields.

[11] presents a hardware solution in a Virtex-4 FPGA coded in RTL encompassing the components Ethernet, IP, UDP, and FAST protocol engine. A microcode engine with an instruction set and a compiler to support different trading protocols are also shown. The component responsible for FAST decoding works at 160MHz, achieving latency of 2.6us for the Network Interface Controller (NIC) and the FAST decoding components altogether.

[12] provides a domain-specific accelerator for market data, along with a FAST decoder, a network interface, and a PCIe host. They adopted an architecture based on an FSM-coordinated Sequence Mapping Table (SMT), which enables the conversion of the FAST templates into programmable control words for the hardware. The FAST decoder engine achieves 300MHz, with a latency of 0.18 - 0.226us.

[13] presents a hardware accelerator for FAST protocol decoder. The architecture consists of a preprocessing of the FAST data to split the PMAP and user data to send to the following stages of the pipeline, which produces signals to control the fields to be decoded. The component achieves a frequency of 125MHz with a latency of 1.6us per message on average.

To sum up, most of the mentioned works aim to explore the flexibility and the processing speed provided by FPGA devices to accelerate one or more layers of a high-frequency trading system. In the same sense, our proposed hardware component implements a B3 template used in production featuring 59 fields, which has more fields than most of the previous solutions, affecting the decoder, as seen in the [10]. As a solution, the necessary computation demanded by the fields is hard-coded in each necessary state to avoid communication bottleneck, and the results achieved by our work are comparable to those with smaller templates.

4 Implementation

The FAST decoder was implemented in both hardware and software. The software implementation used as a golden model is presented in Section 4.1 while the hardware in Section 4.2. The hardware validation occurs through an RTL testbench used to generate stimulus and to collect the results, which is depicted in Section 4.3. In this work, we provide a solution that can receive the FAST data stream and process the messages in a three-stage pipeline, where each stage holds a Finite State Machine (FSM) as explained in Section 4.2.

4.1 Golden Model

To validate the proposed hardware component, a raw message log of FIX/FAST messages from B3 FTP is used. A software solution was developed to mimic the hardware component, which is used as a golden model to compare results from both implementations.

The template 145, known as MDIncRefresh_145, is used to decode the market data for incremental order book changes, in real-time. The software has also the functionality to convert the B3 FAST log to canonical FIX messages, which allows a comparison between the software output and the B3 equivalent FIX log file. With this parser validated, the order book handling function could be validated as well.

4.2 Hardware Component

The three FSMs of the hardware component performs the following actions. The first FSM loads the bytes from a FAST message and identify the necessary template engine. The second FSM is an engine responsible for identifying the fields of the message from template 145, and the third FSM outputs the necessary tags for order book update.

Figure 1 shows the whole hardware component, where each FSM is represented by a diagram sub-block. The readMessage FSM has the states to read the technical header, the PMap, and the TemplateID fields. After reading these fields, it sends all the data to the MDIncRefresh_145 FSM, that when it reaches its last stage, it sends the result fields to the orderBookUpdates FSM.

The readMessage FSM diagram is represented in Fig. 2. The state transitions occur when the MSB, that indicates the stop bit, is one and a flag named end_field, that indicates when the data of the field is processed, is then set to one. The *Wait1* add a delay of one clock cycle to wait the data to be available from the input ports after a start is performed. The readHeader state reads the first ten bytes from the message, and split them into the four corresponding fields of the unencoded technical header of the message. Only the MsgSeqNum and MsgLength fields. The next state decodes the PMap field, which indicates the subsequent fields that depend on the respective bit in the PMap vector. The template ID decoder is the next state, which can be acquired from the message stream if the correspondent PMap bit is set to one. The controlDecoder state checks if the template ID exists and if it is equal to 145, then it verifies if this template decoder engine is available by checking the flag *MDIncRefresh_145*. Then it starts to send all the remaining payload bytes to the decoder machine, which will be available by the MDIncRefresh_145 in the next clock cycle. It keeps reading the message payload field by field until the end of the message

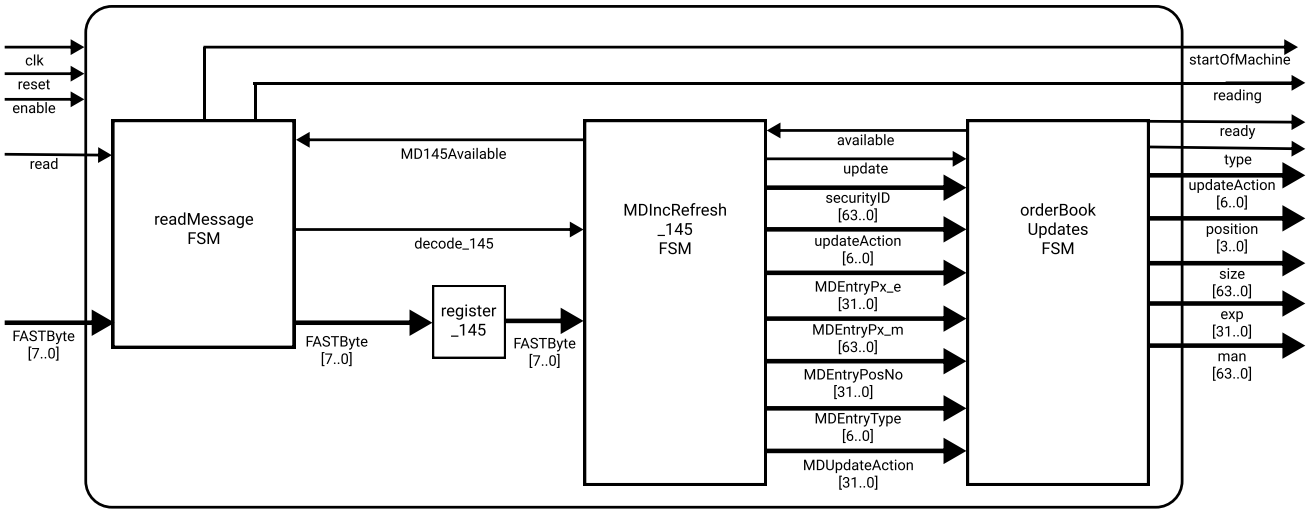

Figure 1 Top view of the FAST hardware component. It is a three stage pipeline, where each stage represents a block of the diagram that holds a finite state machine.

is reached, which is known by comparing the loop trip count with the MsgLength field.

The MDIncRefresh_145 FSM is the main part of the hardware, which is responsible to decode a FAST message for the Template 145. The FSM diagram is depicted in Fig. 3. Each state identifies a specific field: the MsgSeqNum, the SendingTime, the TradeDate, the NoMDEntries, and the MDSequence. The MDSequence is a group of states which represents the MDEntries sequence. A total of 50 states exist to decode up to 50 possible fields from the messages. For example, to decode a MDEntryPositionNo value, which is an uInt32, optional, with a PMap bit associated (the bit of position 21), with no operator, with any initial value and a nullable field, it executes the algorithm depicted on Fig. 4.

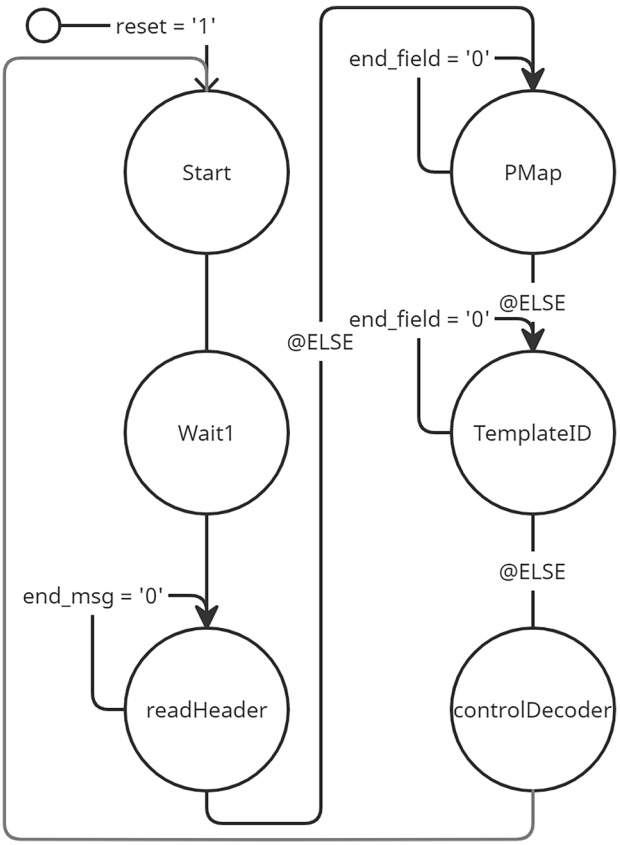

Figure 2 Read Message FSM.

The orderBookUpdates FSM has the objective to receive the fields needed to build the order book, and outputs all the fields needed to update it. The output signals implemented for an order book update are correspondent to the tags MDEntryType, MDUpdateAction, MDEntryPositionNo, MDEntrySize, and MDEntryPx (exponent and mantissa).

The proposed component is controlled by the following signals available through its interface. To start decoding a message, the component needs of one clock cycle with the *reset* set to one, in order to set the outputs *startOfMachine* to be one, and *reading* to be zero. Then, the input *read* needs to be set to one. The component can receive a byte in each clock cycle conditioned to the *reading* port being set to one. Then, the *startOfMachine* is set to zero and the *reading* is set to one. When the *startOfMachine* returns to 1, the component is ready to receive another message. To handle the order book, the testbench used as a wrapper for the component under test needs to read the *type*, *updateAction*, *position*, *size*, *exp*, and *mant* ports, always when the *ready* port is set to one. These are the information required to update an order book. Figure 5 demonstrates an order book update performed by the testbench. In the example, the *ready_out* signal indicates that there is an order book

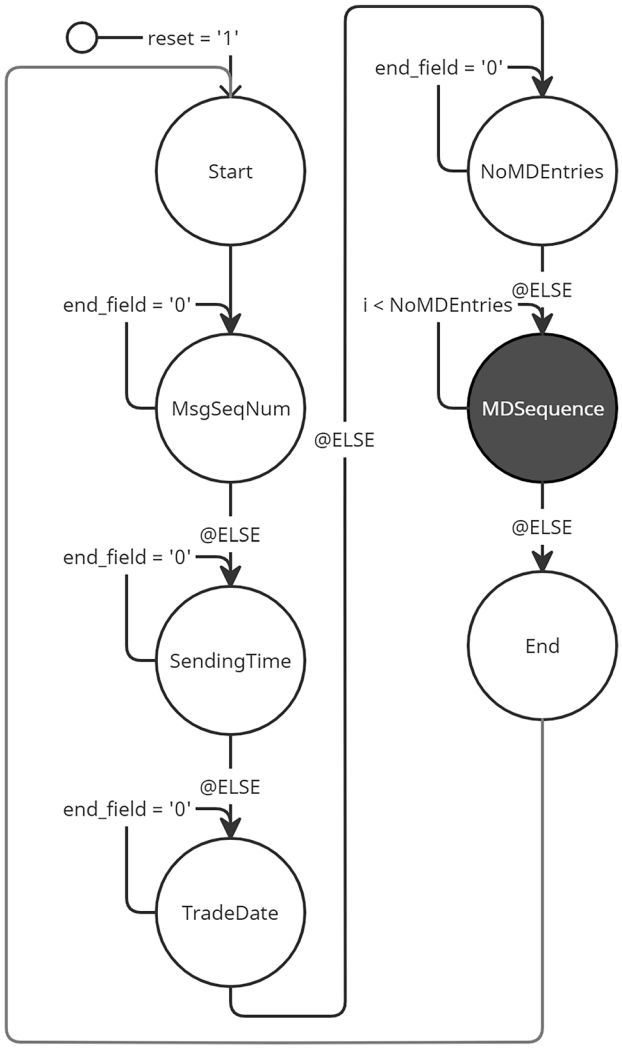

Figure 3 MDIncRefresh_145 FSM.

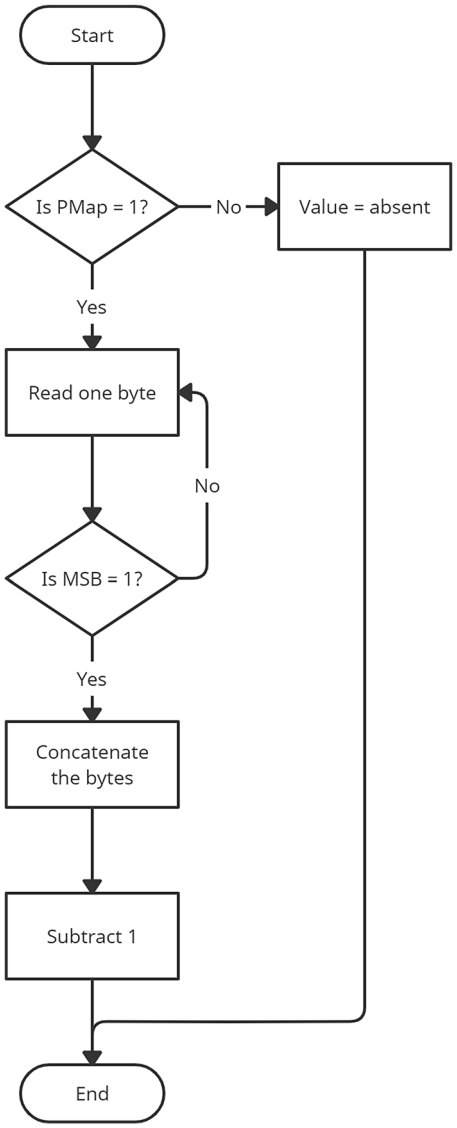

Figure 4 Flow char of a uInt32 field decoder.

update. The *updateAction_out* informs the market data update action, which is from a type *new*. The *type_out* points that it is a bid insertion. So, the testbench inserts the data in the corresponding position and shifts the other positions as necessary. The result for the Fig. 5 is an order book insertion with a new line at position 1 (index 0) of the bid size (*bidOrderDepthBook_size* signal), adding volume 300 at price BRL 30 (*bidOrderDepthBook_px_exp* and *bidOrderDepthBook_px_man* signals). Shifting the previous data at this position to position 2 (index 1), as seen in the volume 1400 at price BRL 26.46 in the new line, which is possible to see in the second clock cycle.

All these three FSMs run in parallel, while the MDIncRefresh is processing some byte, the readMessage reads the next one and orderBookUpdates sends the commands to update the order book. The FSM MDIncRefresh_145 can be replaced to implement different templates. For example, Fig. 6 shows the circuitry from the MDIncRefresh_145 block in more detail. The datapath is composed of several hardware components, each one responsible for a template field decoder. The components are selected by a demultiplexer according to the FSM state. When the last component of the datapath is reached, the commands for the order book are generated and sent to the next FSM, that is the orderBookUpdates.

4.3 Testbench

The mentioned in the previous section, a testbench was used to wrapper the Device Under Test (DUT). The testbench sends FAST log to the hardware component and collects the commands to build an internal order book. It consists of another FSM that starts to write byte by byte in the input

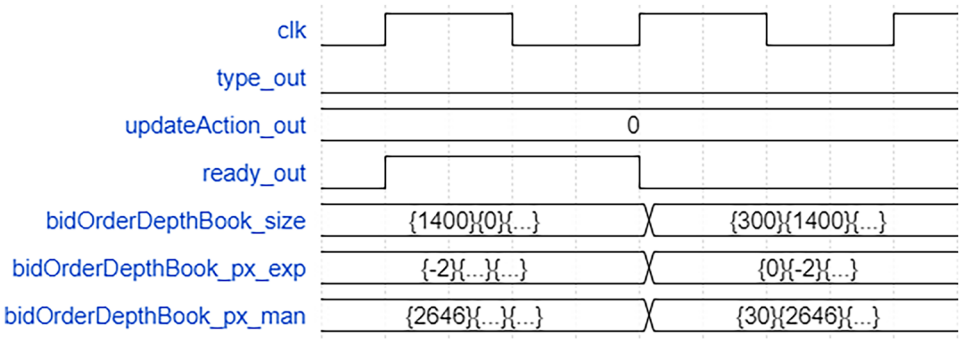

Figure 5 Waveform diagram showing an order book update.

port of the DUT while the *reading* output port is 1. As soon as the *ready* output port of the DUT is set to 1, the testbench reads its output ports in order to get the necessary commands to perform an order book update. Then, it manages the book according to the commands to have an internal order book order by price and grouped by volume, also known as Market By Price (MBP).

Then, a comparison with the results from the software and from the testbench could be conducted. It was possible to check that the order book handled from the software was identical to the one produced by the testbench.

5 Results

Table 3 lists the hardware resources needed to implement the component in a Stratix V 5SGXEA7K2F40C2 FPGA, operating at 164MHz. The average latency to read, decode and outputs the order book fields is 0.72us per message. The results are for a template of 59 fileds long over a set of 59 messages with average length equal to 85 bytes.

The software version used as a golden model was also evaluated in relation to performance, which has achieved average latency of 209us per message, including the input operations necessary to read the bytes from the log file. It was evaluated by counting clock cycles 30 times for each message of the log and taking its averages. The software was not parallelised and it was run in a single core of the processor Intel(R) Xeon(R) CPU E5-1607 3.10 GHz. The strong data dependency of the protocol prevents to have the benefit of parallelism at the core level.

Table 2 compares the related works with our proposed solution. As can be seen, our solution has better latency than [11] and has equivalent latency to [10], however, it has worse latency than [3].

The proposed work presents an equivalent performance compared to the work [10], and better results than [11]. The [3] the achieves a lower latency due to creating its own FAST template. The [9] also achieves a lower latency due to its pipeline optimization. By normalizing the latency according to the template size, work [3] and [9], which are the fastest ones, has a latency of 16.4ns and 17-20ns per template field, respectively, while our solution has latency of 12.2ns per field. When compared to our software golden model, the FPGA solution has a reduction in latency by two orders of magnitude.

The design was optimized in order to have its resources reduced. So, the signals that store the fields were reduced according to the characteristics of the messages of the vendor, which could be seen through the log. For example, the field MDEntryType is of type string, but the values it can receive, defined by the dictionary, are all one-byte length.

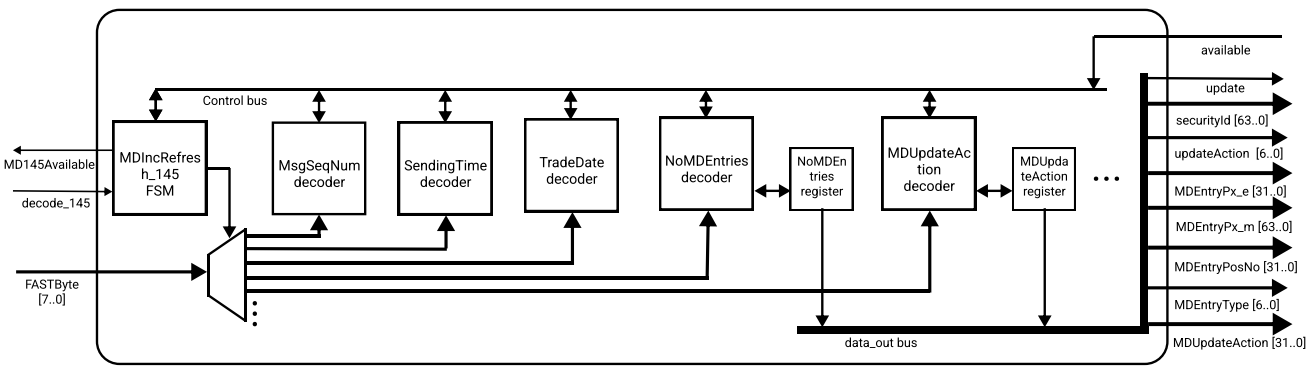

Figure 6 Template 145 decoding circuit.

Table 2 Table of comparison among related works

Work	Latency (μzs)	Latency/field (ns)	Design Method	Template No. of fields	Msgs. Sizes (bytes)
[12]	0.18 - 0.226	17 - 20	RTL	9, 11, 12 and 21	18 - 45
[10]	0.5 - 1.3us	55 - 61	HLS	9, 11 and 21	Not given
[3]	0.115	16.4	HLS	7	Max. of 16
[11]	2.6	-	RTL	Not given	Not given
[13]	0.76	15.2	RTL	50	Not given
This work	0.72	12.2	RTL	59	85 on average

Table 3 Stratix V FPGA resources used

Resource	Used	Available	Utilization (%)
LUT (ALM)	3147	234720	1
FF	354	938880	0
DSP	0	256	0
Block RAM	0	6.25MB	0

It was possible to achieve better performance while fewer recourses were needed. The work [14] did it similarly, as its system considers string fields up to 16 bytes, since a string field in FAST is rarely bigger than this size in a practical scenario. Another way to better design the hardware resources is by checking the FAST dictionary.

6 Possible integration into HFT System

HFT systems are highly customised computational systems used for both to access and process market data and to place orders to the market. HFT differentiates from traditional systems by providing ultra low latency solutions, where specific algorithms (trading robots) can take advantages of such speed by better exploiting the order book liquidity and also to work with multiple orders at the same time, such as arbitration. Even though latency can be mitigated in both software and hardware, solutions in hardware are highly desired since they can provide not only extremely low latency but also better processing time determinism. FPGAs are usually adopted for such solution, as their reconfiguration capability helps to adapt the hardware according the communication protocols and to the investor negotiation strategies [3, 11, 15, 16].

Figure 7 presents the main blocks of a typical HFT system. When implemented in hardware, those blocks can be organised as kernels to work in stream mode at network level proving round-trip latency in the order of nanoseconds. The hardware component presented in this paper can be used as a Feed Decoder (FIX/FAST) to decode messages specific for the B3 templates to be then forwarded to the Order Book block. In this work we consider that the UDP payload is provided to the FASTByte[7..0] input interface and the OrderBook block understands the commands generated by the output interface.

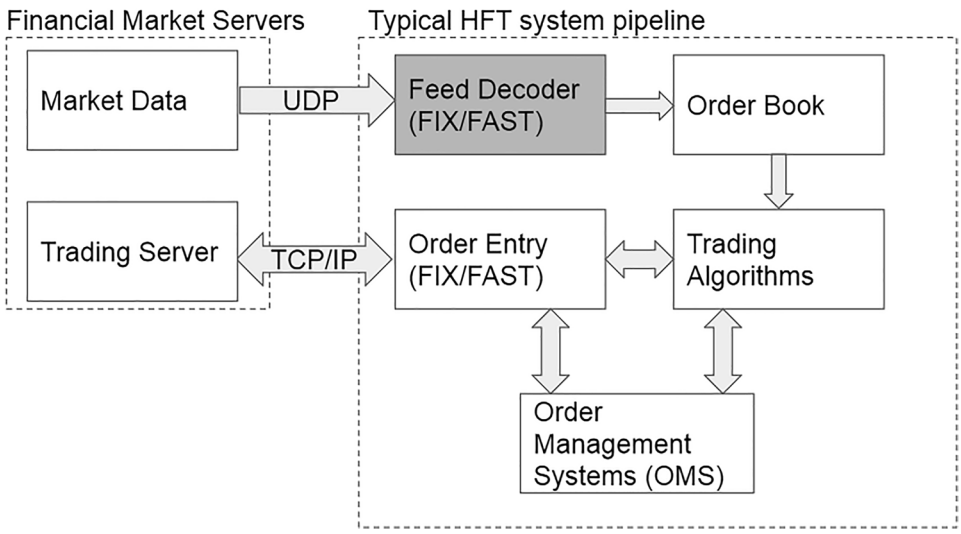

Figure 7 Block diagram of a typical HFT system, where UDP is used for market data feed and TCP/IP for order entry. The Feed Decoder decoders the market data, while Order Entry block places orders to the market according to orders generated by the Trading Algorithm block. Order Book block is the database where real-time market data is stored.

7 Conclusion

The proposed hardware can process FAST messages with the number of chunks equals to one, with template ID 145 (MDIncRefresh), which defines the necessary fields to update the order book. The results are equivalent to the related works, and to the best of our knowledge, this work is the first to propose a FAST engine for the B3 template. The proposed component can be associated with other components in order to build an HFT system applied to financial applications. The average latency of each message from the log is 0.72us.

The use of FPGA-based hardware is often requested due to its low latency implementations. In this work we have implemented the most complicated template, which is responsible to extract the commands to keep order books updated by its incremental changes. The stream-orientated approach from our implementation, where each FSM can be seen as an independent kernel in this stream, could achieve state-of-the-art performance. The proposed IP can be used as a component embedded in a hardware pipeline when a Software Defined Network approach is adopted, which can bring a great benefit to HFT systems since the cost of transferring data from the network level to the host main memory can be avoided.

This hardware component is available as open source at https://github.com/caiossoliveira/FASTEngine-IP, which can be expanded to support other templates as well. Most of the construction and optimizations proposed in this work can be reused, which can speed up further developments. Such components are usually provided as commercial solutions. The golden model software can be accessed in https://github.com/caiossoliveira/FASTEngineMD.

Acknowledgements The authors would like to thank CAPES for the financial support given to this research project.

References

1. Kohda, S., & Yoshida, K. (2021). Characterists of high-frequency trading and its forecasts 2021. *IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1496-1501.
2. Aldridge, I. (2013). *High-frequency trading: a practical guide to algorithmic strategies and trading systems, 306*. Canada: Jihn Wiley & Sons.
3. Boutros, A., Grady, B., Abbas, M., & Chow, P. (2017). Build Fast, Trade Fast: FPGA-based high-frequency trading using high-level synthesis. *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1-6.
4. FIXTRADING. (2017). What is FIX?. www.fixtrading.org/what-is-fix/
5. Li, H., Fu, Y., Liu, T., & Wang, J. (2014). Fast protocol decoding in parallel with fpga hardware. *IEEE 17th International Conference on Computational Science and Engineering*, pages 1669–1672.
6. Lockwood, J. W., Gupte, A., Mehta, N., Blott, M., English, T., & Vissers, K. (2012). A Low-latency library in fpga hardware for high-frequency trading (HFT). *IEEE 20th Annual Symposium on High-Performance Interconnects* pages 9–16.
7. B3 FIX/FAST Message Reference. (2022). www.b3.com.br/pt_br/solucoes/plataformas/puma-trading-system/para-desenvolvedores-e-vendors/umdf-sinal-de-difusao/
8. FIXTRADING, FAST Specification (2006). www.fixtrading.org/packages/fast-specification-version-1-1/
9. Hua, D., Ren, J., Liu, C., & Santhanam, R. (2013). Hardware accelerated decoding of fix/fast and book building of market data, CSEE 4840 Spring 2013, pages 1–183.
10. Tang, Q., Su, M., Jiang, L., Yang, J., & Bai, X. (2016). A scalable architecture for low-latency market-data processing on FPGA, 2016 *IEEE Symposium on Computers and Communication (ISCC)*, pages 1–7.
11. Leber, C., Geib, B., & Litz, H. (2011). High frequency trading acceleration using FPGAs. *21st International Conference on Field Programmable Logic and Application*, pages 317–322.
12. Jia, H., Yuxiang, H., Ding, C., Yan, Y. Cui, J., Wang, J., Cai, C., Xu, L., Zou, Z., & Zheng, L. A. (2022). Domain-specific accelerator for ultra-lowlatency market data distribution system. *IEEE Transactions on Industrial Informatics*, pages 1–11.
13. Zhou, L., Jiang, J., Liao, R., Yang, T., & Wang, C. (2015). FPGA Based low-latency market data feed handler, communications in computer and information science, V. 491, Springer, pages 69–77.
14. Yu, L., Fu, Y., & Liu, T. A.(2017). Hardware structure for FAST protocol decoding adapting to 40gbps bandwidth. *3rd International Conference on Computer Science and Mechanical Automation* (CSMA 2017), pages 290–296.
15. Denholm, S., Inoue, H., Takenaka, T., Becker, T., & Luk, W. (2015). Network-level FPGA acceleration of low latency market data feed arbitration. *IEICE Transactions on Information and Systems, V. E98.D, N.* 2, pages 288–297.
16. Kao, Y. -C., Chen, H. -A., & Ma, H. -P. (2022). An FPGA-Based High-Frequency Trading System for 10 Gigabit Ethernet with a Latency of 433 ns. *International Symposium on VLSI Design, Automation and Test* (VLSI-DAT), pages 1–4.