Received 28 June 2022, accepted 12 July 2022, date of publication 27 July 2022, date of current version 1 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3194265

RESEARCH ARTICLE

Multi-COBS: A Novel Algorithm for Byte Stuffing at High Throughput

ENRICO RONCONI, (Member, IEEE), NICOLA CORNA, (Member, IEEE), ANDREA COSTA, (Member, IEEE), FABIO GARZETTI, (Member, IEEE), NICOLA LUSARDI, (Member, IEEE), AND ANGELO GERACI, (Senior Member, IEEE)
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, 20133 Milan, Italy

Corresponding author: Enrico Ronconi (enrico.ronconi@polimi.it)

ABSTRACT Framing methods are used to break a data stream into packets in most digital communications. The use of a reserved symbol to denote the frame boundaries is a popular practice. This end-of-frame (EOF) marker should be removed from the packet content in a reversible manner. Many strategies, such as the bit and byte stuffing processes employed by high-level data link control (HDLC) and Point-to-Point Protocol (PPP), or the Consistent Overhead Byte Stuffing (COBS), have been devised to perform this goal. These bit and byte stuffing algorithms remove the reserved EOF marker from the packet payload and replace it with some extra information that can be used to undo the action later. The amount of data added is called *overhead* and is a figure-of-merit of such algorithms, together with the encoding and decoding speed. Multi-COBS, a new byte stuffing algorithm, is presented in this paper. Multi-COBS provides concurrent encoding and decoding, resulting in a performance improvement of factor four or eight in common word-based digital architectures while delivering an average and worst-case overhead equivalent to the state-of-the-art. On the reference 28-nanometer field programmable gate array (FPGA) (Artix-7), Multi-COBS achieves a throughput of 6.6 Gbps, instead of 1.7 Gbps of COBS. Thanks to its parallel elaboration capability, Multi-COBS is ideal for digital systems built in programmable logic as well as modern computers.

INDEX TERMS Byte stuffing, FPGA, framing, packet, transmission.

I. INTRODUCTION

Most digital communications are nowadays performed using packet-oriented protocols. As an example, communication over the largest computer network, *Internet*, is performed using the Internet Protocol (IP, either version 4 [1] or 6 [2]) which is packet oriented. This is also true on a smaller scale, for example over point-to-point protocols like PCI-Express [3] or Universal Serial Bus [4] (USB). A packet-oriented protocol organizes the data in units called *packets* which are formatted and typically composed by control information and the actual data that should be transmitted, the *payload* [1].

Multiplexing multiple transmissions on the same physical link (e.g., by inserting a tag in the control data [1]), implementing an error detection mechanism (e.g., by inserting a checksum in the control data [5]), recovering communication after an error (since the corruption on one packet usually does not affect subsequent packets), implementing a flow-control mechanism, and so on are all advantages of organizing data in packets.

Most digital communication physical links transmit data represented in bits, often grouped in octet (bytes) or in multiple of 8 bits. This introduces a fundamental problem: since both the payload and the control data are represented using bits, and since data can be arbitrary, a packet-oriented protocol should implement some mechanism to distinguish between the actual payload and the control information added [6].

A common technique, used for example by the aforementioned IPv4, is to reserve a fixed number of bytes at the beginning and at the end of the packet, which contains the control information. However, to implement such technique,

The associate editor coordinating the review of this manuscript and approving it for publication was Byung-Seo Kim.

it is necessary to provide some mechanism to separate the packets, i.e. to mark where a packet ends and where the next begins. Such mechanism is called *framing* [1].

Many of the currently used framing techniques, introduced in the Section § II, have been developed in the last decades and are tailored to be efficiently executed on microcontroller and microprocessor based architectures, addressing almost the entire use cases in that period of time. For this reason, those algorithms can not be easily parallelized to exploit the concurrent computing capabilities of modern programmable logic (PL) devices.

On the other hand, nowadays an increasing amount of applications (e.g, [7]–[9]) require the communication between programmable logic devices and a microcontroller or microprocessor. Common scenarios include PL devices that send data to a workstation where they are further elaborated [8], [9] or PL devices that can be configured from software running on a computer [7]. For such reason, a wide number of communication frameworks for PL-microprocessor links have been recently developed [10], [11]. Such systems rely on framing techniques that should be capable of being executed on both PL and microprocessor-based devices.

In this paper, a novel framing algorithm called *Multi-COBS* is presented. The Multi-COBS is based on the existing Consistent Overhead Byte Stuffing [12] (COBS) algorithm and can be used to frame packets using an end of frame marker. While COBS can only elaborate one byte every cycle, Multi-COBS has overcome this constraint and can now elaborate multiple bytes per cycle, resulting in a significant increase in encoding and decoding throughput. Multi-COBS was created to have efficient encoding and decoding implementations in both programmable logic and software, and the reference implementation given in Section § VI boosts throughput from 1.7 to 6.6 Gbps in a 32-bit system and 13.2 Gbps in a 64-bit system. This remarkable increment of performance is provided at the cost of slightly higher latency and algorithm complexity with respect to current state of art framing techniques.

First, a brief description of COBS is given, then in Section § IV the Multi-COBS algorithm is described and analyzed in terms of performance from a theoretical point of view. In Section § V is described and analyzed a possible PL implementation, then the experimental performance of such implementation is analyzed in Section § VI.

II. STATE OF THE ART

For encoding variable-sized frames, there are a number of well-known algorithms. Prepending the payload length before each frame, for example, is a simple technique that allows the decoder to simply separate the received transmission into the various frames when sent over an ideal communication channel. However, it is clear that this technique is unsuitable for real-world applications: data can be lost, distorted, or spurious one can appear during transmission [13], and if the payload size varies or the prepended data becomes garbled, the receiver would never be able to recover its decoding process. Furthermore, because the payload length must be prepended, such a solution necessitates knowing the precise data size prior to the start of the transmission, which necessitates buffering the entire payload and causes unpredictable latency.

Reserving a special code termed an *end-of-frame (EOF) marker*, which will be put after each frame to signal its end, is a more resilient framing mechanism [14]. In some situations, such as the transmission of ASCII-encoded text over a byte-oriented channel, the EOF marker can be set to a byte value that never appears in the data, ensuring that the marker byte is solely used to mark the frame end. When transmitting arbitrary data, however, it is not possible to select a reserved value that will not appear in the payload.

To overcome this difficulty, various methods have been developed: given a specified reserved code (the EOF marker), they erase all occurrences of that code from the data and add some extra information that will subsequently be used to rebuild the original payload. The length of such excess information is referred to as *overhead*, and it is one of the algorithms' figures of merit.

In paragraph II-A HDLC and PPP algorithms are presented, while in II-B the COBS is briefly described and the three algorithms are compared in II-C.

A. CURRENT BIT- AND BYTE-STUFFING ALGORITHMS

The so-called bit-stuffing and byte-stuffing methods are procedures for removing a reserved code from a data stream in a reversible manner. They both work on the same principle: whenever the reserved code is detected in the input data, it is updated at the bit or byte level, and extra data is crammed near the code to keep track of the changes [15].

For example, in the high-level data link control [14] (HDLC) protocol, packets are delimited using the byte `0x7E` = `0b01111110` and the delimiter is removed from the payload using the following bit oriented procedure: when five consecutive ones occurs in the data, an extra zero bit is stuffed right after the fifth one. Since the EOF marker contains six consecutive ones, but the data is modified inserting a zero after a sequence of five ones, the EOF maker does not appear in the encoded data. To undo this operation, the decoder discards every zero bit that follows a sequence of five ones, while if a sequence of six ones followed by a zero is met, it is interpreted as frame terminator. Using this protocol, the decoder will never receive more than six consecutive ones except in case of errors.

Bit-stuffing algorithms are efficient in terms of overhead, but they are difficult to implement in software on modern processors, which often require byte (or multiple byte) registers, operations, and memory access. When single bits are filled, padding to byte is required; additionally, detecting a bit sequence that spans multiple bytes requires multiple operations, whereas a simple byte comparison normally just requires one.

TABLE 1. Minimum, maximum, and average overheads for PPP, HDLC, and COBS [12]. Among the algorithms shown, COBS has the lowest worst-case and average overhead.

	Min	Max	AVG
HDLC	0	20.0%	1.61%
PPP	0	100.0%	0.78%
COBS	1 byte	0.4%	0.23%

Instead, other protocols like Point-to-Point Protocol [16] (PPP) use a byte oriented procedure to remove the frame delimiter from the input data. The PPP framing algorithm defines two special values, the EOF marker `0x7E` (the same used by HDLC) and a so-called *escape* byte `0x7D`. In order to remove all EOF marker occurrences from the data, two substitutions are performed: each `0x7E` byte is replaced by the two bytes sequence `0x7D 0x5E` and each `0x7D` byte is replaced by the two bytes sequence `0x7D 0x5D`. Each substitution will insert a two bytes sequence that starts with the escape character. To undo this operation, the decoder should take action every time `0x7D` is received: the escape character should be discarded, and the following byte should be replaced according to the scheme `0x5E` $\rightarrow$ `0x7E` and `0x5D` $\rightarrow$ `0x7D`.

The presented bit and byte stuffing algorithms adds a variable overhead: they can add no overhead at all if the reserved characters never appears in the input data, but in the worst case the overhead can be up to 20% in case of HDLC (since five bits are encoded using six bits) and 100% in case of PPP (since every byte is encoded using two bytes) [12].

B. CONSISTENT OVERHEAD BYTE STUFFING

The COBS [12] is another algorithm used to remove a reserved code from a stream. COBS main feature is the limited worst case overhead: 0.4% of payload size (for payloads with length $\gg 254$). The algorithm is extensively described and analyzed in [12] and the following is just a brief illustration of its encoding and decoding procedure, needed for the description of the Multi-COBS presented in the next sections. COBS works in bytes, thus it's expected that the reserved code to remove is zero (`0x00`) in the following. To delete any code from the stream, the same approach can be used with small tweaks or data pre-processing (e.g., apply a bitwise XOR between input bytes and the desired reserved code).

1) COBS ENCODING

To begin, a `0x00` byte is prepended to the input data, and a "virtual" `0x00` is appended. As the last encoding process, the appended virtual zero is dropped; it is not required, but it simplifies the subsequent explanation. The actual implementation does not save it in memory but acts as if it is present.

After this operation, starting from the first byte (the prepended zero) and proceeding to the end of data, each zero byte in position i is replaced with the value $v = \min(k - i, 255)$ where k is the position of the next zero byte. Moreover, if $v = 255$, an extra `0x00` byte is stuffed in position $i + 255$. Such stuffed `0x00` byte will be then substituted using the same rules. The last appended zero should not be substituted; instead, it should be discarded as the final action. Figure 1 depicts the encoding of an example sequence.

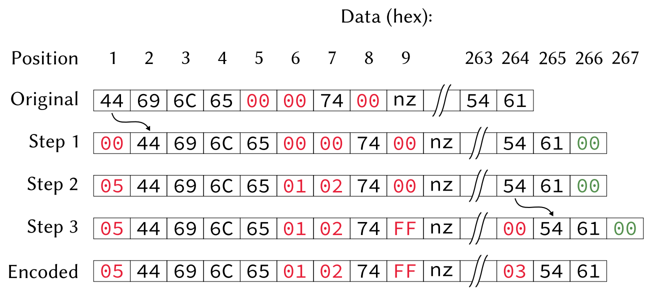

Data (hex):
Position 1 2 3 4 5 6 7 8 9 263 264 265 266 267
Original 44 69 6C 65 00 00 74 00 nz 54 61
Step 1 00 44 69 6C 65 00 00 74 00 nz 54 61 00
Step 2 05 44 69 6C 65 01 02 74 00 nz 54 61 00
Step 3 05 44 69 6C 65 01 02 74 FF nz 00 54 61 00
Encoded 05 44 69 6C 65 01 02 74 FF nz 03 54 61

FIGURE 1. Example of COBS encoding. The input sequence is 264 bytes long and has 254 non-zero bytes from position 9 to 264 included (in figure, `nz` indicates non-zero bytes). Step 1 prepends and appends zero bytes to the original data, step 2 replaces the first three zeros, step 3 replaces the zero in position 9 with 0xFF and stuffs an extra zero byte in position 264. Finally, the zero in position 264 is replaced, the appended zero now in position 267 is discarded, and the encoding is complete. The original data was 264 bytes in length, whereas the encoded data is 266 bytes long. The overhead for this payload is 2 bytes, or 0.76 percent of the original payload size.

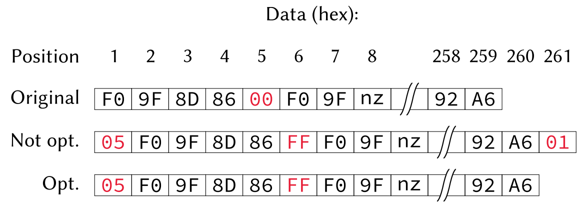

Data (hex):
Position 1 2 3 4 5 6 7 8 258 259 260 261
Original F0 9F 8D 86 00 F0 9F nz 92 A6
Not opt. 05 F0 9F 8D 86 FF F0 9F nz 92 A6 01
Opt. 05 F0 9F 8D 86 FF F0 9F nz 92 A6

FIGURE 2. Optimization of a COBS specific case. The first line displays the input data, which includes the last zero at position 5, or 254 bytes before the payload ends. The encoding without the exception for optimizing frames is shown in the second line, which ends with a chunk of exactly 254 non-zero bytes. The third line shows the identical payload encoded with such optimization, which saves one byte.

A small exception can be added to the described encoding procedure to slightly decrease average and worst case overhead: if the last `0x00` byte to be encoded is located exactly 254 bytes before the end of payload, it will be substituted with value 255 accordingly with the previous paragraph. However, in this scenario, an extra `0x00` byte is not required. Figure 2 depicts an example of this optimization's functioning.

2) COBS DECODING

This three-step cyclic operation is performed until there are no more bytes to process to decode a COBS message:

1) The first byte in the decoding queue is removed from the queue and its value x stored in an auxiliary register;
2) $x - 1$ bytes are removed from the decoding queue and put in the output queue;
3) If $x < 255$ then a `0x00` byte is put in the output queue.

C. COMPARISON

Table 1 shows the contrast between the given HDLC (bit-oriented) and the byte-oriented PPP and COBS. COBS was created to restrict the worst-case overhead and cuts it from

100% of PPP to 0.4 percent. COBS not only lowers the worst-case scenario, but also lowers the average overhead from 0.78 percent of PPP to 0.23 percent. COBS, on the other hand, introduces a minor increase in latency [12].

III. MULTI-COBS STRATEGY

Multi-COBS is a novel byte stuffing technique developed to overcome the throughput bottleneck of one byte per clock cycle introduced by COBS in modern systems. This section gives an overview of the problem and possible trivial solutions for concurrent framing (III-A). Then, Multi-COBS encoding and decoding procedures are described respectively in paragraphs paragraphs III-B and paragraph III-C.

Next sections will then analyze the novel algorithm from a theoretical point of view (Section § IV) and experimental benchmarks are presented in paragraph paragraph VI-B.

A. PARALLEL FRAMING

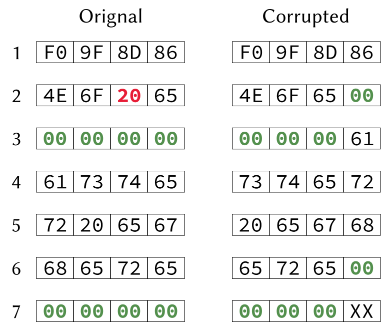
Multi-COBS was created with the goal of framing data sent across a bidirectional point-to-point communication channel between an FPGA and an x86_64 workstation. For this application, an algorithm with fairly efficient encoding and decoding routines that is straightforward to implement in both software and hardware should be identified. For reasons described in II-A, the algorithm should not be bit-oriented, whereas for programmable logic implementation, an approach that can be easily parallelized is more suited to better use the spatial computing capabilities of such devices. Furthermore, a minimal worst-case overhead saves resources such as buffers, which should always be allocated to accommodate the worst-case scenario in hardware, even if it is extremely unusual.

COBS has a low overhead, but it only processes single bytes and cannot be easily parallelized, so a hardware encoder's throughput can't exceed one byte per clock cycle. On the other hand, typical in-chip communication buses such as Advanced eXtensible Interface [17] (AXI) or Avalon [18] are synchronous parallel buses with a data width of usually 32 or 64 bits, to allow higher data rate without increasing the clock frequency. In such a design, a byte-oriented COBS encoder would function as a bottleneck unless it was clocked at a frequency at least four or eight times higher than the parallel bus (respectively for 32 and 64 bits wide buses), which is not always practicable.

COBS might possibly be extended to work with larger words rather than bytes. For example, using n-bytes wide elements, COBS can be modified to prepend and append a n-bytes zero-word to the input sequence and then replace each i-th zero-word with $v = \min(k - i, 2^{8n} - 1)$ obtaining an encoding similar to COBS.

However, such solution would require buffering, in the worst case, $2^{8n} - 2$ elements which is usually not feasible for width of 32 an 64 bit.

Moreover, if such stream is then transferred over a byte-oriented protocol, such as USB 3 or RS 232, if a single byte is lost and thus the words misaligned, the decoding can not be recovered since all the following EOF marker will be corrupted, as shown in Figure 3.



	Orignal	Corrupted
1	F0 9F 8D 86	F0 9F 8D 86
2	4E 6F 20 65	4E 6F 65 00
3	00 00 00 00	00 00 00 61
4	61 73 74 65	73 74 65 72
5	72 20 65 67	20 65 67 68
6	68 65 72 65	65 72 65 00
7	00 00 00 00	00 00 00 XX

FIGURE 3. Example of fatal framing corruption when transmitting zero-word-terminated word sequence over a non-ideal byte-oriented link. On the left part, is shown the original sequence of seven 32-bit word, one word per line. There are two frames composed respectively of two and three words, both terminated with the word 0x0000 0000. Right column shows the same sequence after the red byte has been lost during the transmission. All following EOF marker (in green) are corrupted.

B. MULTI-COBS ENCODING

Multi-COBS works with streams of words, each of which is made up of n concatenated bytes. From a theoretical standpoint, Multi-COBS operates on bytes sequences with a length that is an integer multiple of n, but because the encoder will most likely be supplied via an n-bytes wide parallel bus in practice, it is generally more easy to think of it as a word-based method. Multi-COBS accepts an n-bytes-wide words sequence of length l as input and produces an n-bytes-wide words sequence of length $L > l$ with no zero words (e.g. `0x0000 0000`) as output. To address the byte alignment issue outlined in III-A and Figure 3, Multi-COBS ensures that each full byte of the output sequence is not `0x00`.

The output sequence contains all the information needed to revert this operation and re-obtain the original input sequence. The constraint that both the input and the output sequences have a length in bytes that is multiple of n is important, because it allows both the input and the output to be transferred on a n-bytes wide parallel bus.

Considering n-bytes-wide input sequence $\mathrm{I} = w_i$ of length l, where the i-th word can be expressed as concatenation of n bytes $w_i = b_i^0 b_i^1 \cdots b_i^{n-1}$, it is possible to write the sequence as

$$\begin{aligned} \mathrm{I} &= (w_0, w_1, \cdots, w_{l-1}), & (1)\\ &= \left(b_0^0, b_0^1, \cdots, b_0^{n-1}, b_1^0, b_1^1, \cdots, b_{l-1}^{n-2}, b_{l-1}^{n-1}\right). & (2) \end{aligned}$$

The Multi-COBS algorithm works by slicing the input sequence into n sub-sequences of bytes $\mathrm{S}_0, \mathrm{S}_1, \cdots, \mathrm{S}_{n-1}$ according to the scheme

$$\mathrm{S}_i = \left(b_0^i, b_1^i, \cdots, b_{l-1}^i\right). \tag{3}$$

Take a look at Figure 4 for an example of visually depicted slicing.

After this operation each sub-sequence S_i is encoded independently using the COBS algorithm. E_i will represent

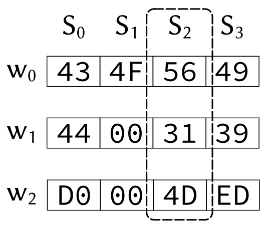


FIGURE 4. Multi-COBS input sequence slicing example. The input sequence composed of 32-bits words I = (0x434F 5649, 0x4400 3139, 0xD000 4DED) is visually represented by writing one word per line. Since the input words are composed by 32 bits i.e. 4 bytes, the input sequence is sliced in 4 sub-sequences $S_0, \cdots, S_3$. The sequence S_2 = (0x56, 0x31, 0x4D) has been highlighted with a dashed box to make it more clear.

the COBS-encoded sequence associated with S_i. Despite S_i sequences being all l-bytes long, the COBS encoded sequences potentially have different lengths since the COBS encoding introduces an overhead that is dependent on its input data. Each sub-sequence E_i is then padded – if needed – to the same length by appending one or more `0x01` bytes at the end of each sequence, except the longest one. Adding `0x01` at the end of the encoded sequence maintains the COBS encoding valid: such bytes will be interpreted by the decoder as extras zeros. This last padding operation will guarantee that all the E_i sequences have now the same length L, thus the total number of bytes in such sequences is an integer multiple of n and we can find a way to merge them into a single word sequence.

All the bytes sub-sequences $E_i = \left(B_0^i, B_1^i, \cdots, B_{L-1}^i\right)$ are merged into the Multi-COBS output sequence $O = (W_0, W_1, \cdots, W_{L-1})$, where each word is defined as the concatenation $W_i = B_i^0 B_i^1 \cdots B_i^{n-1}$ and the Multi-COBS encoding process is complete. Since all the sub-sequences E_i do not contain any `0x00` byte (thanks to COBS encoding), by construction it can be seen that output sequence do not contain any word composed by only zero bytes (i.e. $W_i \neq 0$ for each $0 \leq i < L$).

C. MULTI-COBS DECODING

The Multi-COBS decoding technique begins by slicing the input sequence in the same way as the encoding phase. The inverse COBS method is then used to decode each sub-sequence separately. Some sub-sequences may have been padded during the encoding phase by appending one or more `0x01` bytes, resulting in an extra zero at the end of the decoded sequence. However, since the encoded sub-sequences have been padded to match the longest one, there is always at least one encoded sub-sequence which have not been padded and thus will be decoded to its original form. Extra zeros can then be discarded by trimming all the decoded sub-sequences to the length of the shortest one.

Finally, decoded and trimmed sub-sequences can be merged into a unique sequence of words as it has been previously described for the encoding phase.

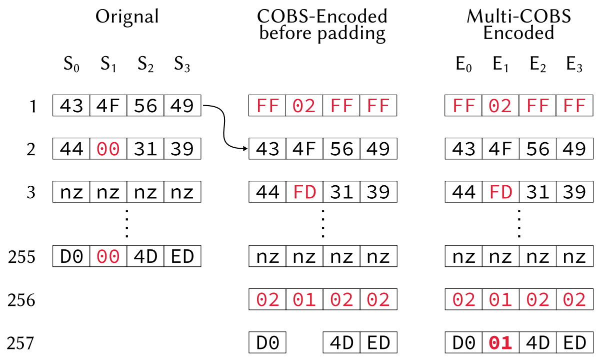

Orignal
COBS-Encoded before padding
Multi-COBS Encoded
S0 S1 S2 S3
E0 E1 E2 E3
1 43 4F 56 49 | FF 02 FF FF | FF 02 FF FF
2 44 00 31 39 | 43 4F 56 49 | 43 4F 56 49
3 nz nz nz nz | 44 FD 31 39 | 44 FD 31 39
255 D0 00 4D ED | nz nz nz nz | nz nz nz nz
256 | 02 01 02 02 | 02 01 02 02
257 | D0 4D ED | D0 01 4D ED

FIGURE 5. Multi-COBS encoding example. The input 32-bit word sequence is shown on the left side, one word per line for a total of 255 words. Vertical columns represent the sub-sequences S_i and `0x00` bytes in each sub-sequence are highlighted in red. In the middle part, each sub-sequence has been encoded with COBS. Sub-sequences E_0, E_1 and E_2 are 257 bytes long while the sequence E_1 is 256 bytes long. According to Multi-COBS encoding, the sequence E_1 has been padded to 257 bytes by appending `0x01` (in bold) and the final result is shown on the right part of the image. The output, on the right, is in fact a 32-bit-words sequence with length 257 (two words of overhead). The decoding procedure of the same data is shown in Figure 6.

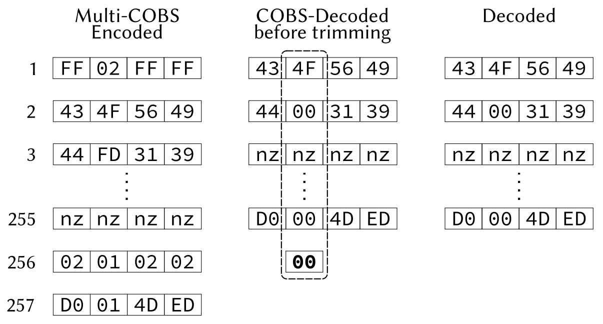

Multi-COBS Encoded
COBS-Decoded before trimming
Decoded
1 FF 02 FF FF | 43 4F 56 49 | 43 4F 56 49
2 43 4F 56 49 | 44 00 31 39 | 44 00 31 39
3 44 FD 31 39 | nz nz nz nz | nz nz nz nz
255 nz nz nz nz | D0 00 4D ED | D0 00 4D ED
256 02 01 02 02 | 00
257 D0 01 4D ED

FIGURE 6. Multi-COBS decoding example. It is represented the encoded sequence previously utilized in the Figure 5 example on the left side. In the middle, the inverse COBS algorithm was used to decode each encoded sub-sequence separately. Notice that, due to earlier padding during the encoding method, an extra zero (in bold) has been inserted to the sub-sequence in the dashed rectangle. Finally, the longest sequence is shortened, and the decoded payload is displayed on the right side, which is identical to the original.

D. COBS AND MULTI-COBS COMPARISON

Because each sub-sequence can be encoded or decoded independently of the others, Multi-COBS uses n COBS encoders/decoders that handle data concurrently. Multi-COBS should, in theory, guarantee an n-fold increase in throughput over COBS encoding/decoding. In practice, some additional logic is necessary to execute the padding and trimming, thus the actual throughput may be lower than anticipated. In VI-B, real-world benchmarks are reported.

E. FRAMING

Multi-COBS removes all the occurrences of the code 0 from a words sequence. After this operation, the 0 word can be used as a separator to mark the end of each words sequence when they are sent over a serial medium.

Moreover, it is clear from the previous sections that after Multi-COBS encoding not only zero words are not present, but each word does not contain any zero byte, i.e. if $W_i = B_i^0 B_i^1 \cdots B_i^{n-1}$ then

$$B_i^j \neq \texttt{0x00} \text{ for each } 0 \leq i < l \text{ and } 0 \leq j < n, \tag{4}$$

where l is the length of the output word sequence. It is possible to take an advantage of this property if the framing of the encoded sequence should be sent over a byte-oriented link. In this case, it is not necessary to insert a whole zero word to separate the frames, but a single zero byte is sufficient since it is guaranteed that it will not appear in the payload. This also addresses the byte misalignment problem previously presented in III-A.

Multi-COBS can be also used in case a lower-level byte framing is not available. Until now, it has been assumed that the underlying medium is capable of transmit whole bytes and that bit-to-byte alignment is never broken during the transmission, but this may not be guaranteed over some serial protocols. In this case, we can again exploit the property (4): a Multi-COBS encoded stream does not contain more than fourteen consecutive zeros (in case of adjacent `0x80 0x01` bytes) thus it is possible to use a one followed by fifteen zeros i.e.,

$$\texttt{0b1000 0000 0000 0000},$$

to mark the end of frame without ambiguity.

IV. THEORETICAL ANALYSIS

It is interesting to evaluate the Multi-COBS algorithm in terms of overhead (i.e. the size of the extra information added) and to analyze how it compares with COBS and with the aforementioned HDLC and PPP solutions. The best, worst, and average overhead for both COBS and Multi-COBS are examined in the following sections.

A. BEST-CASE OVERHEAD

1) COBS

When it comes to COBS, a byte is always prepended, regardless of the input data. The best case is represented by a payload in which zero bytes are so frequent that no sequence of 254 or more non-zero bytes is present. In this case, the prepended byte is the only byte added to the sequence. Thus, the best case overhead for COBS is one byte, regardless of the payload size.

2) MULTI-COBS

When a sequence of l words is encoded using Multi-COBS, if and only if each sub-sequence COBS encoding incurs in the best-case overhead described in the previous paragraph, each sub-sequence increases its length by exactly one byte.

In this case, all COBS encoded sub-sequences have a length of $l + 1$ bytes and are merged in the output word sequence without padding. Thus, in this case the output sequence is exactly $l + 1$ words long and the overhead is one word (n bytes).

This represents the best-case overhead of Multi-COBS: even if a single sub-sequence grows by $x > 1$ bytes, then all sub-sequences will be padded to $l + x$ bytes and the output word sequence will have length $l + x > l + 1$ words.

B. WORST-CASE OVERHEAD

1) COBS

After prepending a byte, the COBS algorithm adds an extra overhead byte for each sequence of 254 non-zero bytes followed by another byte. If the 254 non-zero sequence is not followed by another byte (i.e. if it is the last payload byte) then the overhead byte it is not inserted. The worst case is then represented by an input sequence composed only by non zero bytes.

If the worst case payload is shorter than 255 byte (including no-payload frames) the overhead is exactly one byte.

If the worst case payload has a length multiple of 254 bytes, $l = 254 \cdot m$, the overhead will be $1 + m - 1$. The first $+1$ is due to the prepended byte, while the last -1 is due to the fact that the last 254 non-zero byte sequence will not cause the insertion of an extra byte.

In general, for a payload that has a length $l > 0$ that is not an integer multiple of 254 bytes, the overhead is $1 + \left\lfloor \frac{l}{254} \right\rfloor = \left\lceil \frac{l}{254} \right\rceil$. The following is a summary of the above:

$$\text{COBS overhead}_{\text{wc}} = \begin{cases} 1 & \text{if } l = 0 \\ \left\lceil \dfrac{l}{254} \right\rceil & \text{if } l > 0 \end{cases} \tag{5}$$

$$= \left\lceil \frac{l}{254} \right\rceil \approx \frac{l}{254} \quad \text{if } l \gg 254 \text{ bytes} \tag{6}$$

that is a worst-case overhead of $1/254 \approx 0.4\%$ of the payload size for sufficiently large frames ($l \gg 254$ bytes).

2) MULTI-COBS

Multi-COBS has the worst-case overhead if at least one COBS-encoded sub-sequence has the worst overhead. All sub-sequences will be padded to match the longest one, which will reach overhead of (expressed in words)

$$\text{Multi-COBS overhead}_{\text{wc}} = \begin{cases} 1 & \text{if } l = 0 \\ \left\lceil \dfrac{l}{254} \right\rceil & \text{if } l > 0 \end{cases} \tag{7}$$

$$\approx \frac{l}{254} \quad \text{if } l \gg 254 \text{ words.} \tag{8}$$

This shows how the Multi-COBS worst-case overhead is in fact the same of COBS: $1/254 \approx 0.4\%$ of the payload size for sufficiently large frames ($l \gg 254$ words).

C. AVERAGE OVERHEAD

1) COBS

COBS average payload-relative overhead for long packets ($l \gg 254$ bytes) has been calculated in [12]:

$$\frac{\left(\sum_{i=1}^{254} i \cdot \mathrm{p}(i)\right) + 255 \cdot (255/256)^{254}}{\left(\sum_{i=1}^{254} i \cdot \mathrm{p}(i)\right) + 254 \cdot (255/256)^{254}} - 1, \tag{9}$$

where the function $\mathrm{p}(x)$ is defined as the probability of receiving a sequence of $x-1$ non-zero bytes followed by a zero:

$$\mathrm{p}(x)=\left(\frac{255}{256}\right)^{x-1}\cdot\frac{1}{256}.$$

The average overhead is equal to 0.2295% of the payload size.

2) MULTI-COBS

The Multi-COBS overhead expressed in words is $x=\max\left(e_0,e_1,\cdots,e_{n-1}\right)$, where e_i is the overhead (the **e**xtra size) of the COBS-encoded i-th sub-sequence, expressed in bytes. The max () operator takes into account the padding operation that is performed by Multi-COBS encoder. While best- and worst-case overhead are the same for COBS and Multi-COBS, we expect higher average overhead for Multi-COBS, because the Multi-COBS overhead is determined by the sub-sequence that expanded the most during encoding.

The expected overhead x of Multi-COBS encoding of a sequence with l words (each word n-bytes long) is then:

$$\bar{x}=\mathrm{E}\left[\max\left(e_0,e_1,\cdots,e_{n-1}\right)\right]. \tag{10}$$

The probability that Multi-COBS overhead is equal or less than j is

$$\mathrm{P}\left(x\leq j\right)=\left[\mathrm{P}\left(e\leq j\right)\right]^n, \tag{11}$$

because each encoded sub-sequence must have an overhead of less than or equal to j for this to happen. We also know that $\mathrm{P}\left(x\leq 0\right)=0$ since there is at least one word of overhead and also $\mathrm{P}\left(x\leq\left\lceil\frac{l}{254}\right\rceil\right)=1$, since the overhead can never exceed the worst case overhead calculated in IV-B2.

With this information it is possible to evaluate

$$\mathrm{P}\left(x=j\right)=\mathrm{P}\left(x\leq j\right)-\mathrm{P}\left[x\leq\left(j-1\right)\right]. \tag{12}$$

Finally, the expected Multi-COBS overhead can be calculated by definition as

$$\mathrm{E}\left[x\right]=\sum_{j=1}^{j=\left\lceil\frac{l}{254}\right\rceil} j\cdot\mathrm{P}\left(x=j\right). \tag{13}$$

However, one crucial component is missing: in order to calculate the average Multi-COBS overhead the probability distribution of COBS overhead $\mathrm{P}\left(e\leq j\right)$ should be known. While the average COBS overhead for a random input sequence has been calculated in [12] (0.2295% of the payload size), no information about its probability density is available.

3) COBS OVERHEAD PROBABILITY DISTRIBUTION

We can model the encoder as a finite-state machine (FSM) to calculate the probability distribution of COBS encoding for an input sequence of l bytes. The FSM has l states $q_0,q_1,\cdots,q_{l-1}$. Our goal is to count how many overhead bytes are added, thus we'll look for 254 non-zero bytes followed by additional byte sequences. Each state does so by keeping track of two values: the current size of overhead and the number of consecutive non-zero bytes received.

Let's encode such a state with pair (a,b):

$$(a,b)=q_{255(a-1)+b}. \tag{14}$$

Before receiving the first byte, the system starts in the state $q_0=(1,0)$ since we have already accumulated an overhead byte, but no non-zero byte have been encoded. After processing the first byte, the state may have changed to $q_1=(1,1)$ if a non-zero byte has been processed, or remained in q_0 if processed a zero byte. When starting from state q_1, processing another byte can cause a jump from q_1 to q_2 (if byte is non-zero) or to q_0 (if byte is zero). The pattern is repeated until q_{254} is reached, in particular for all $0\leq i<254$ from the state q_i two jumps are possible: toward q_{i+1} with probability $^{255}/_{256}$ or toward q_0 with probability $^{1}/_{256}$. When the FSM reaches state $q_{254}=(1,254)$, any other encoded byte (either zero or non-zero) will increase the overhead to two bytes. In fact, if another byte is encoded after the state q_{254}, a sequence of 254 non-zero bytes followed by another byte has been encoded. Moreover, if in state q_{254} a zero byte is received, the state will change to $(2,0)=q_{255}$, otherwise the received non-zero byte is potentially the first byte of the next non-zero sequence, leading to a jump to state $(2,1)=q_{256}$.

It is possible to summarize as follows: for each state q_i where $i\neq 254+k\cdot 255\,\forall k\in\mathbb{N}$ it is possible to jump either to state q_{i+1} if non-zero byte is received, or to jump back to state $q_{255\cdot\lfloor i/255\rfloor}$ if a zero byte is received:

$$\mathrm{P}\left(q^{n+1}=q_{i+1}|q^n=q_i\right)={}^{255}/_{256}, \tag{15}$$

$$\mathrm{P}\left(q^{n+1}=q_{255\cdot\lfloor i/255\rfloor}|q^n=q_i\right)={}^{1}/_{256}. \tag{16}$$

Moreover, for each state q_i where $i=254+k\cdot 255\,\forall k\in\mathbb{N}$ it is possible to jump either to state q_{i+1} if a zero byte is received, or to state q_{i+2} if a non-zero byte is received:

$$\mathrm{P}\left(q^{n+1}=q_{i+1}|q^n=q_i\right)={}^{1}/_{256}, \tag{17}$$

$$\mathrm{P}\left(q^{n+1}=q_{i+2}|q^n=q_i\right)={}^{255}/_{256}. \tag{18}$$

It can be noticed that the probability of the next state q^{n+1} depends only on the current state q^n: the model satisfies the Markov property.

It is possible to calculate the transition square-matrix T where each element $t_{i,j}$ (located at i-th row and j-th column) is defined as the probability of jumping to state q_j from state q_i:

$$t_{i,j}=\mathrm{P}\left(q^{n+1}=q_j|q^n=q_i\right). \tag{19}$$

It follows that the elements of T^n represents the probability, starting from i, of ending up being in state j after n jumps:

$$\mathrm{P}\left(q^n=q_j|q^0=q_i\right)=\left(T^n\right)_{i,j}. \tag{20}$$

In case of the COBS encoder, the starting state is always $q^0=q_i$, so the only interesting part is the first row of matrix T^n which represents the probability of ending up in each state

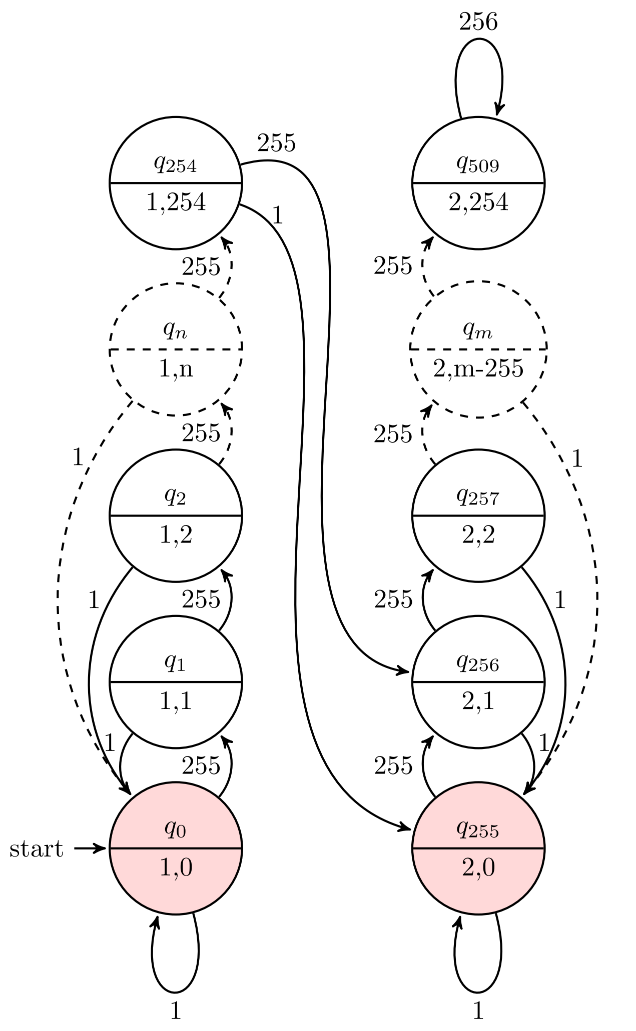


FIGURE 7. Modeling the encoding of up to 508 bytes with COBS encoding using a state diagram. The pair (current overhead bytes, non-zero counter) is reported at the bottom of each state. This diagram can model the encoding of up to 508 bytes, for this reason the state will never exceed q_{509}. So, to respect the law of total probability, there is only one (never executed) jump with probability 1 from state q_{509} to q_{509}. In the left column are represented states with only one byte of overhead, on the right state with two bytes of overhead.

after encoding n bytes. Summing up the first 255 elements (from q_0 to q_{254}) of such row will result in the probability of having a single byte of overhead; summing up the first 510 elements (from q_0 to q_{509}) will result in the probability of having an overhead less than or equal to two bytes, etc. In general, the probability of overhead less than j bytes is:

$$P(e \leq j) = \sum_{i=0}^{255 \cdot j - 1} \left(T^n\right)_{0,i}. \tag{21}$$

During real calculations, a matrix T large enough to include all the attainable states should be employed. It can be verified that after encoding l bytes, the state can not exceed $q_{\left\lfloor \frac{l-1}{254} \right\rfloor + l}$ so it is sufficient to construct the T matrix with a size $\left(\left\lfloor \frac{l-1}{254} \right\rfloor + l + 1\right) \times \left(\left\lfloor \frac{l-1}{254} \right\rfloor + l + 1\right)$. Calculation of

TABLE 2. Relative average overhead (%) for the same number of bytes encoded with COBS or Multi-COBS, rearranged in 32 or 64 bit words. The minimal extra overhead introduced by Multi-COBS is more than compensated by its throughput gain.

Payload size [bytes]	COBS	Multi-COBS	
	8 bit	32 bit	64 bit
1	100.00	–	–
4	25.00	100.00	–
8	12.5	50.0	100.0
256	0.54	1.56	3.12
512	0.37	0.78	1.56
1024	0.29	0.72	0.78
2048	0.26	0.48	0.72
4096	0.25	0.37	0.48
8192	0.24	0.32	0.37

average overhead of COBS for sequences with finite length of l-bytes are reported in Table 2 along with calculation, using equations (11) and (13), of Multi-COBS expected overhead. From such table it can be seen that Multi-COBS maintains the overhead (for large packets) well beyond 0.5% while offering a remarkable throughput.

Figure 8 reports the overhead probability distributions calculated for a fixed stream length for both COBS and Multi-COBS (32 and 64 bit). As expected, the best and worst case are equal when the same number of "elements" (words or bytes) are encoded, but the average overhead is shifted towards the worst case in Multi-COBS. Wider words are sliced in a higher number of sub-sequences which will be padded to the longest one, leading to higher average overhead. In fact, the plot shows that 64 bit Multi-COBS have higher average overhead than 32 bit Multi-COBS.

D. LATENCY

The somewhat significant latency during the encoding process is one disadvantage that Multi-COBS inherited from COBS. In fact, Multi-COBS should wait until the first encoded byte is available for all n encoded sub-sequences before producing the first encoded word. However, to produce such bytes it is necessary to analyze – in the worst case of all non-zero sub-sequences – up to 254 words.

COBS and Multi-COBS have substantially higher latency than HDLC, which can produce each output bit by reading only one bit, or PPP, which can produce each output byte by reading only one byte. A summary of all figures of merit of PPP, HDLC, COBS and Multi-COBS is reported in Table 3.

On the other hand, Multi-COBS decoding does not require lookahead an thus does not introduce latency.

As previously stated, the slightly higher latency is the drawback of Multi-COBS with respect to HDLC and PPP. However the 254 clock cycles latency translates to less than 1 μs latency if the encoder is clocked at 400 MHz (maximum

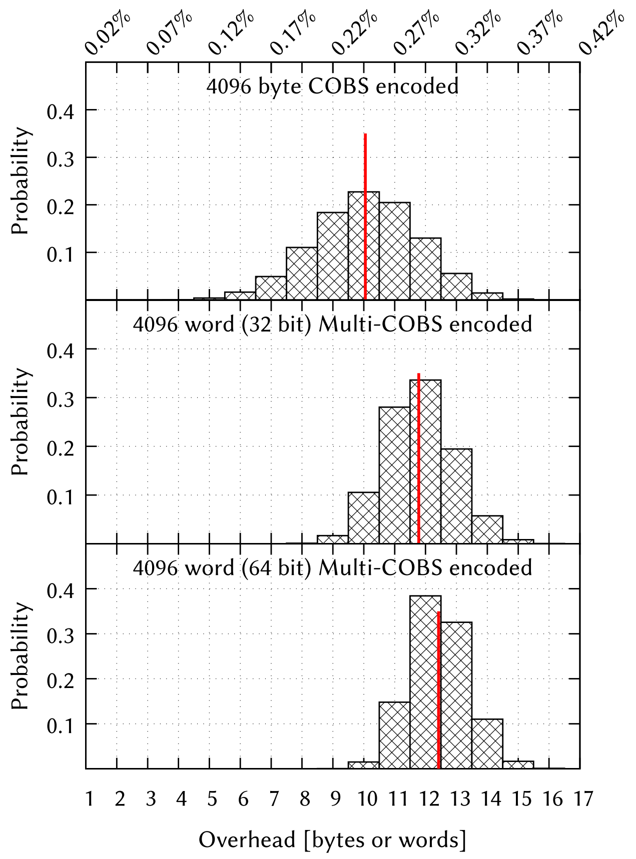


FIGURE 8. Overhead probability distribution when encoding 4096 bytesusing COBS or 4096 words using Multi-COBS (with 32 bit and 64 bit words). The vertical red line represents the average of the distribution mean, that is the expected value of overhead.

TABLE 3. Overhead comparison. Even if Multi-COBS overhead is slightly larger than COBS, it performs very well with respect to PPP and HDLC in both average and worst case.

	Max latency	Overhead		
	(encoding)	Min	Avg	Max
HDLC	0	0	1.61%	20.0%
PPP	0	0	0.78%	100.0%
COBS	254 cycl	1 byte	0.23%	0.4%
Multi-COBS (32 bit)	254 cycl	1 word	0.32%	0.4%
Multi-COBS (64 bit)	254 cycl	1 word	0.37%	0.4%

working frequency will be later discussed in VI-B). Such small latency is often negligible with respect to the physical medium latency. Moreover, many applications where a large quantity of data should be transmitted, benefit from an increased throughput regardless of the introduced latency.

V. COBS AND MULTI-COBS HARDWARE IMPLEMENTATION

COBS algorithm have been public for more than twenty years and many hardware [19] and software [20] implementations are available. COBS or COBS-like protocols have been used successfully in several programmable logic architectures [21], [22].

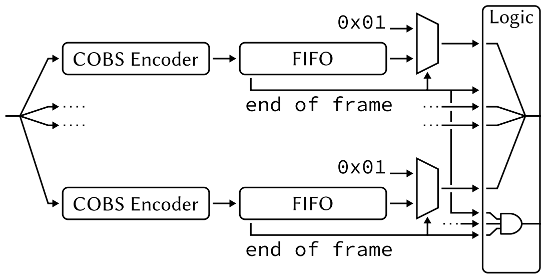


FIGURE 9. Working principle of the Multi-COBS hardware encoder.

Multi-COBS encoding and decoding is relatively easy to implement if a COBS encoding/decoding logic is already available.

A. MULTI-COBS ENCODING

By splitting the parallel bus into multiple 8-bit parallel buses, the slicing operation may be accomplished easily in hardware. Then, in parallel, n COBS encoders can be created, each synchronized with the same clock as the input parallel bus. The bytes should be concatenated back into words after this stage. To produce the output word, n bytes from each COBS encoder must be read in parallel to form the concatenation operation. Because the COBS overhead varies by sub-sequence, a FIFO between the COBS encoders and the "concatenation" stage is required to hold the variation in sub-sequence overhead over time.

In particular, each FIFO should be large enough to accommodate the difference between the sub-sequence with less overhead and the one with most, which in the worst case is (from Equation 7 and IV-A):

$$\text{COBS overhead}_{\text{wc}} - \text{COBS overhead}_{\text{bc}} = \left\lceil \frac{l}{254} \right\rceil - 1. \tag{22}$$

Moreover, when a COBS encoder detects the end of stream, a multiplexer injects the byte `0x01` to pad each encoded sub-sequence to the same length. When all COBS encoders signal end of stream, the Multi-COBS encoder has finished encoding the current frame and the EOF is asserted.

B. MULTI-COBS DECODING

Multi-COBS decoding is quite similar to the encoder: n COBS decoders are instantiated and process the encoded stream in groups of n bytes per clock cycle. Also in this case, the logic has to take into account the padding of each encoded sub-sequence: when at least one COBS decoder detects the end of frame all other bytes before the EOF are discarded from each sub-sequence.

VI. EXPERIMENTAL RESULTS AND PERFORMANCE

To benchmark Multi-COBS performance, the presented architecture have been successfully implemented on Xilinx's Artix-7 and Kintex Ultrascale FPGAs both with speed grade -2. In the following paragraphs are reported the

TABLE 4. **COBS and Multi-COBS encoder ($n = 4$) resource utilization on `xc7a100tftg256` FPGA. BRAMs storage capacity is 36Kbits on the selected architecture.**

		Max payload length		
		16 KiB	128 KiB	1 MiB
Multi-COBS	LUTs	818	900	919
	Regs	426	481	498
	BRAMs	0	2	2
COBS	LUTs	164		
	Regs	54		
	BRAMs	0		

TABLE 5. **COBS and Multi-COBS decoder ($n = 4$) resource utilization on `xc7a100tftg256` FPGA. BRAMs storage capacity is 36Kbits on the selected architecture.**

Multi-COBS	LUTs	443
	Regs	334
	BRAMs	0
COBS	LUTs	50
	Regs	30
	BRAMs	0

FPGA resource utilization and maximum speed in different configurations.

A. FPGA RESOURCES UTILIZATION

The Multi-COBS encoder and decoder have been implemented on Artix-7 FPGA (`xc7a100tftg256-2`) using $n = 4$ and with various maximum payload length. The only parameter influenced by the maximum payload length is the depth of the FIFOs used between the COBS encoder and the word concatenation, as described in V-A. The FPGA resource usage for encoder is reported in Table 4.

In this table, it can be observed that for small payload sizes, no Block RAM (BRAM) tiles are required because the FIFOs are implemented using LUTs, however for bigger payload sizes, the synthesizer uses two BRAM tiles to create the requisite FIFOs. Due to the logic required to implement the FIFO and padding logic, Multi-COBS encoders require 25% to 40% more LUTs and 97 percent to 130 percent more registers than four COBS encoders.

However, if compared to the total resource available in the reference Artix-7 FPGA, the implementation is quite compact as it uses only 1.24% of the LUTs available, 0.39% of the Registers available and 1.48% of the Block RAM available with maximum payload size of 1 MiB.

The FPGA resource use for the Multi-COBS and COBS decoders is shown in Table 5.

It can be seen how the Multi-COBS decoder requires 122% more LUTs and 179% more registers with respect to four COBS standalone decoders. However, also for the decoder the relative utilization (with respect to total resource available on the reference Artix-7 FPGA) is quite low: 0.60% of the LUTs and 0.27% of the registers.

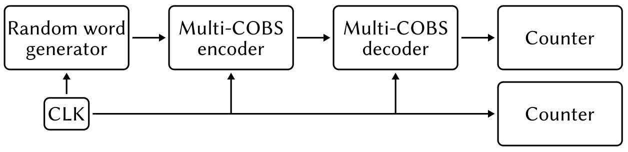


FIGURE 10. **FPGA firmware setup to benchmark the architecture throughput.**

B. MAXIMUM OPERATING FREQUENCY AND THROUGHPUT

On the reference Xilinx's Artix-7 -2 speed grade device, the Multi-COBS encoder have been successfully implemented with a clock frequency of 210 MHz, which allows a theoretical throughput of $6.72 \cdot 10^9$ bit/s when used with $n = 4$, while the relative decoder can be clocked up to 220 MHz.

On a Kintex Ultrascale with speed grade -2 (`xcku040ffva1156-2`) both the encoder and decoder have been successfully implemented with a clock frequency of 400 MHz, which allows a theoretical throughput of $12.8 \cdot 10^9$ bit/s when used with $n = 4$.

To benchmark the actual throughput of the architecture, a Multi-COBS encoder followed by a decoder have been implemented on the reference FPGA, using $n = 4$. With this setup (Figure 10), the average throughput of the pipe when encoding random words is 0.984 words per clock cycles, which translates in 6.6 Gbps on Artix-7 with speed grade -2 and 12.6 Gbps on Kintex Ultrascale with speed grade -2.

Using the same setup but implementing COBS encoder plus decoder offers a throughput of 0.992 words per clock cycles, showing that the actual throughput gain of Multi-COBS with $n = 4$ with respect to COBS is

$$\frac{0.984 \cdot 4}{0.992} = 3.978,$$

very close to the ideal case (factor 4).

Those experimental results in terms of latency and throughput are summarized in Table 6. As predicted theoretically in IV-D, Multi-COBS does not worsen the latency but increases the throughput by a factor of almost 4 (32-bit Multi-COBS) or almost 8 (64-bit).

Considering Multi-COBS encoding of n-bytes wide words, and taking as ideal case the throughput of a single COBS encoder multiplied by n, it can be calculated that with $n = 4$ (32-bit) Multi-COBS reaches 99.19 % of ideal throughput while with $n = 8$ (64-bit) 98.81% of ideal performance is reached, demonstrating that the throughput scales almost linearly with the number of COBS encoder in parallel, with minimal penalty.

VII. REAL APPLICATIONS

Despite the fact that FPGAs are becoming more used in digital systems, connecting them with PCs is problematic due to the lack of standard dedicated peripherals to handle common out-of-chip communication protocols. As a result, numer-

TABLE 6. Multi-COBS and COBS throughput and latency comparison.

	COBS	Multi-COBS	Multi-COBS
		(32 bit)	(64 bit)
Throughput [bytes/cycl]	0.992	3.936	7.842
Max latency [cycl]	254	254	254

ous communication architectures have been devised to carry out this task [23], [24]. When compared to similar COBS, Multi-COBS can be utilized to frame packets exchanged with such systems, providing the aforementioned benefits: very low average and worst-case overhead and high throughput. In fact, [11] designed and documented a communication system that leverages Multi-COBS for framing. A communication system like this was used to connect an x86_64 workstation to a high-performance FPGA-based Time-to-digital converter (TDC) measuring device [9], [25] through multiple physical links such UART, USB 2, and USB 3. In this case, a high-throughput communication link should be available to transport the collected measurement – without loss – to the operator workstation for storage and offline elaboration. Multi-COBS, which provides a high-throughput framing solution, was critical but fundamental in completing this work.

VIII. CONCLUSION

Multi-COBS extends the well-known COBS algorithm to allow it to operate on larger words, delivering a stunning throughput boost of almost four times when operating on 32-bit words and possibly nearly eight times when operating on 64-bit words. The reference implementation raised throughput from 1.7 Gbps of COBS to 6.6 Gbps of Multi-COBS in a 32-bit system and to 13.2 Gbps in a 64-bit system in real-world terms. The additional logic required to expand COBS to Multi-COBS is simple to comprehend and describe, and it can be implemented with minimum FPGA resources (less than 2% on medium-sized FPGAs). When compared to alternative framing techniques employed by HDLC and PPP, which have worst case overheads of 20 percent and 100 percent, Multi-COBS inherits the key feature of COBS: it strictly limits the worst case overhead (0.4% percent for big payloads). Multi-COBS' average overhead is comparable to that of COBS, albeit slightly higher. For these reasons, Multi-COBS is a fascinating approach that is well suited to most FPGA-based applications that demand a low average and worst-case overhead framing algorithm.

REFERENCES

[1] *Internet Protocol*, Internet Engineering Task Force, document RFC 791, Sep. 1981.

[2] S. E. Deering and B. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, Internet Engineering Task Force, document RFC 8200, Jul. 2017.

[3] D. Mayhew and V. Krishnan, "PCI express and advanced switching: Evolutionary path to building next generation interconnects," in *Proc. 11th Symp. High Perform. Interconnects*, 2003, pp. 21–29.

[4] *Universal Serial Bus Specification Rev 1.0*, Jan. 1996.

[5] *IEEE Standard for Ethernet*, IEEE Standard 802.3-2018 (Revision IEEE Standard 802.3-2015), Aug. 2018, pp. 1–5600.

[6] B. T. Doshi, S. Dravida, E. J. Hernandez-Valencia, W. A. Matragi, M. A. Qureshi, J. Anderson, and J. S. Manchester, "A simple data link protocol for high-speed packet networks," *Bell Labs Tech. J.*, vol. 4, no. 1, pp. 85–104, Jan. 1999.

[7] A. Roy, L. Sharma, I. Chakraborty, S. Panja, V. N. Ojha, and S. De, "An FPGA based all-in-one function generator, lock-in amplifier and auto-relockable PID system," *J. Instrum.*, vol. 14, no. 5, May 2019, Art. no. P05012.

[8] D. Holanda Noronha, R. Zhao, J. Goeders, W. Luk, and S. J. E. Wilton, "On-chip FPGA debug instrumentation for machine learning applications," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, New York, NY, USA, Feb. 2019, pp. 110–115.

[9] N. Corna, F. Garzetti, N. Lusardi, and A. Geraci, "Digital instrument for time measurements: Small, portable, high–performance, fully programmable," *IEEE Access*, vol. 9, pp. 123964–123976, 2021.

[10] R. F. Molanes, L. Costas, J. J. Rodriguez-Andina, and J. Farina, "Comparative analysis of processor-FPGA communication performance in low-cost FPSoCs," *IEEE Trans. Ind. Informat.*, vol. 17, no. 6, pp. 3826–3835, Jun. 2021.

[11] N. Corna, E. Ronconi, F. Garzetti, S. Salgaro, N. Lusardi, L. Tavazzani, and A. Geraci, "High-performance physical-independent address-based communication interface for FPGA in custom scientific equipment," in *Proc. IEEE Nucl. Sci. Symp. Med. Imag. Conf. (NSS/MIC)*, Oct. 2020, pp. 1–4.

[12] S. Cheshire and M. Baker, "Consistent overhead byte stuffing," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 159–172, Apr. 1999.

[13] G. Breed, "Bit error rate: Fundamental concepts and measurement issues," *High Freq. Electron.*, vol. 2, no. 1, pp. 46–47, 2003.

[14] *High-Level Data Link Control Procedures (HDLC)—Frame Structure*, Ecma International, Geneva, Switzerland, Standard 40, Jan. 1980.

[15] S. Aviran, P. H. Siegel, and J. K. Wolf, "An improvement to the bit stuffing algorithm," *IEEE Trans. Inf. Theory*, vol. 51, no. 8, pp. 2885–2891, Aug. 2005.

[16] W. A. Simpson, *The Point-to-Point Protocol (PPP)*, Internet Engineering Task Force, document RFC 1661, Jul. 1994.

[17] *AMBA AXI and ACE Protocol Specification*, Version D, Arm, Cambridge, U.K., 2011.

[18] *Avalon Interface Specifications*, Version 17-1, Intel Corporation, Santa Clara, CA, USA, Specification, 2018.

[19] A. Forencich. (Jan. 2022). *Verilog AXI Stream Components Readme*. [Online]. Available: https://github.com/alexforencich/verilog-axis

[20] C. McQueen. (Oct. 2021). *Consistent Overhead Byte Stuffing (COBS)*. [Online]. Available: https://github.com/cmcqueen/cobs-c

[21] A. Adetomi, G. Enemali, and T. Arslan, "Clock buffers, nets, and trees for on-chip communication: A novel network access technique in FPGAs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2017, pp. 219–222.

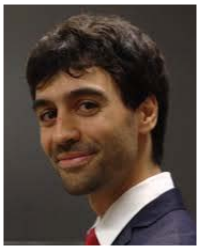
[22] A. Adetomi, G. Enemali, and T. Arslan, "Relocation-aware communication network for circuits on Xilinx FPGAs," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–7.

[23] P. Lieber and B. Hutchings, "FPGA communication framework," in *Proc. IEEE 19th Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2011, pp. 69–72.

[24] N. Alachiotis, S. A. Berger, and A. Stamatakis, "A versatile UDP/IP based PC $\longleftrightarrow$ FPGA communication platform," in *Proc. Int. Conf. Reconfigurable Comput. (FPGAs)*, Dec. 2012, pp. 1–6.

[25] N. Lusardi, F. Garzetti, S. Salgaro, N. Corna, A. Costa, and A. Geraci, "Fully-configurable FPGA-based instrument for multi-channel and multi-histogram time measurements at high-performance," in *Proc. IEEE Nucl. Sci. Symp. Med. Imag. Conf. (NSS/MIC)*, Oct. 2020, pp. 1–4.

ENRICO RONCONI (Member, IEEE) received the bachelor's and master's degrees in electronic engineering from the Politecnico di Milano, in 2017 and 2020, respectively. His research interests include advanced programmable logic (PL) and software architectures for data elaboration and transfer in field programmable gate arrays (FPGA) implemented scientific equipment, currently used and developed together with the time-to-digital and digital-to-time converters (TDC and DTC) developed at the Politecnico di Milano.



NICOLA CORNA (Member, IEEE) was born in 1992. He received the bachelor's and master's degrees in electronics engineering from the Politecnico di Milano, in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree with DEIB, with a focus on the development of systems on FPGA and SoC reconfigurable devices, particularly time-domain devices. He is the author and a developer of various open-source projects. His research interest includes free software.

ANDREA COSTA (Member, IEEE) was born in Piacenza, in 1995. He received the B.Sc. degree in biomedical engineering and the M.Sc. degree in electronics engineering from the Politecnico di Milano, in 2017 and 2020, respectively. His research interests include innovative hardware architectures for data processing in the field of FPGA time-domain devices and FPGA DAQ for high data rate environments.

FABIO GARZETTI (Member, IEEE) received the bachelor's and master's degrees in electronic engineering from the Politecnico di Milano. He developed his thesis work at the Digital Electronics Laboratory, DEIB, on a topic regarding innovative solutions for calibration and triggering of asynchronous signals for time-to-digital converters (TDCs) in field programmable gate arrays (FPGA), where he applied for the award of a temporary research fellowship within the framework of the ''Design of modules for readout and processing of sampled data based on FPGA architectures'' research program, supported by CAEN ELS.

NICOLA LUSARDI (Member, IEEE) was born in Piacenza, in November 1990. He received the Ph.D. degree, in 2018. He developed his thesis work at the Digital Electronics Laboratory, DEIB, on a topic regarding high-resolution time-to-digital converters (TDCs) in field programmable gate arrays (FPGA).

Since 2014, he has been collaborating with CERN in the LHCb experiment, CAEN S.p.A., Viareggio, LU; CAELels S.r.l., Basovizza, Trieste; Elettra Sincrotrone Trieste S.C.p.A., Basovizza; Single Quantum B.V., Delft, The Netherlands; the Technology University of Delft; École Polytechnique Fédérale de Lausanne; and Rete Ferroviaria Italiana. He is now a temporary Researcher at the Digital Electronics Laboratory, DEIB, a Professor of electronics at the Politecnico di Milano, and an Associated Member of the Italian National Nuclear Physics Institute (INFN). His research line and knowledge as a Digital Designer have been acknowledged by public and private research centers. He is the Co-Founder of TEDIEL S.r.l., an Italian start-up and spin-off of the Politecnico di Milano.

ANGELO GERACI (Senior Member, IEEE) received the M.Sc. degree *(cum laude)* in electrical engineering and the Ph.D. degree *(cum laude)* in electronics and communication engineering from the Politecnico di Milano, in 1993 and 1996, respectively. He has been a Scientific Collaborator of the Italian National Nuclear Physics Institute (INFN), since 1995. Since 2004, he has been an Associate Professor at the Department of Electronics, Information and Bioengineering (DEIB), Politecnico di Milano. He is currently a Lecturer for the ''Sistemi Elettronici Digitali'' and ''Digital Electronic Systems Design'' courses at the School of Industrial and Information Engineering, Politecnico di Milano, and he holds courses for the Ph.D. degree Program in information technology. He is a member of the Management Board and the Deputy Coordinator of the Ph.D. School of Information Engineering, Politecnico di Milano. He has been the Advocate and the Manager of several sponsored joint research projects between the Politecnico di Milano and private/public companies. He is an auditor of projects on behalf of the Italian MIUR. He is the author or coauthor of more than 320 publications on refereed international congress proceedings and journals. His research interests include digital electronics based on microcontrollers, DSP, and FPGA devices, specifically in the areas of radiation detection, medical imaging, energy storage for automotive electric systems, and HPC applications. In 2003, he was elevated to the level of a Senior Member of the IEEE Nuclear and Plasma Society. He received the IEEE TRANSACTIONS Prize Paper Award by the IEEE Power Electronic Society, in 2004. As a Referee for several international journals including IEEE TRANSACTIONS ON NUCLEAR SCIENCE and *Review of Scientific Instruments*.