Proposed data format for SPADIC 2.1

Michael Krieger

Revision be0dafb
April 18, 2018

Contents

1. Review of existing SPADIC output data format

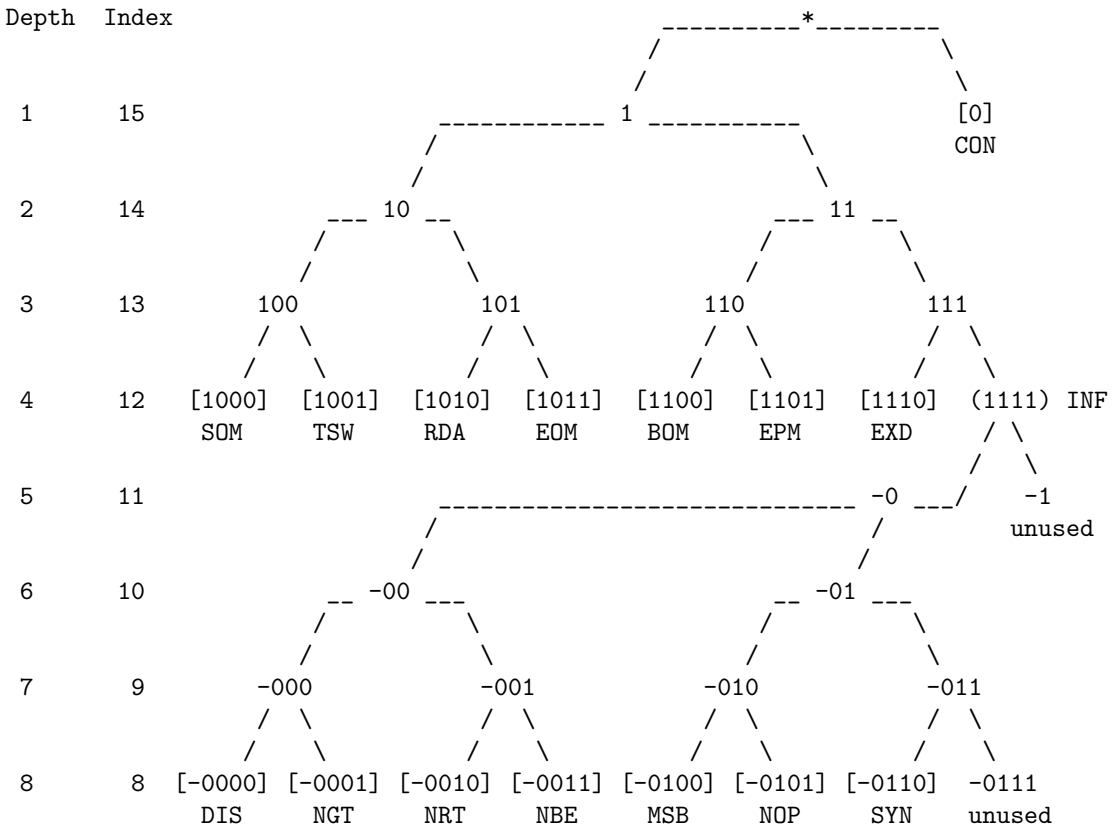
1.1. Words and messages

1.1.1. Word types

In SPADIC 1.0 and 1.1, the data output is formed by a sequence of *words* which are $N = 16$ bits in size. Every word has a *type* which can unambiguously (without taking the context in which the word appears into account) identified by the n-bit *prefix* of the word (i.e., the first n bits from the left), where n depends on the word type. The prefix is also called *preamble* in some documents.

Depending on the type, a word can contain up to $N - n$ bits of data (*payload*), formatted in a type-specific way.

All word types can be arranged in the following tree of all possible prefixes, where all nodes at a given depth n (formatted in a horizontal row) correspond to the prefixes with length n:



The column `Index` indicates the index of the rightmost bit of the prefixes in a given row (where index $15 = N - 1$ corresponds to the leftmost bit of the word), which is equal to the maximum payload size for a given word type.

Prefixes that correspond to a defined word type are formatted using brackets (`[...]`) and the name of the word type written below. The prefixes for all word types must be leaf nodes in the tree, or in other words: no word type can have a prefix which is the ancestor of the prefix of another word type, because that would make determining the word type from the prefix ambiguous.

Note that other documents specify a word type `INF` (*info word*) with prefix `1111`, and several 4-bit *info types* as part of the payload of `INF` words. Within the scope of this document however, these info types are treated as individual word types with prefixes of length $n = 8$, sharing the initial `1111` part of the prefix (which is omitted in the drawing of the tree in order to save horizontal space).

1.1.2. Ignored words

NOP words can be inserted into the sequence of words at any time, and should be ignored by the receiving side. This enables using a transport mechanism which requires sending a minimum number of words in "packets", or sending an *uninterrupted* sequence of words, in situations where there is less actual data to be sent than the word sending rate offered by the transport mechanism would allow.

1.1.3. Messages

With NOP words filtered out, subsequences of the remaining sequence of words can form *messages*. There are several kinds of messages with their own rules of how they are composed:

- Hit messages
- Epoch markers
- Exceptions

During normal operation of the chip, only *hit messages* and *epoch markers* should occur. There are several other kinds of messages that are called *exceptions* in this document, because they should only occur in exceptional cases. If they are seen regularly, it is a sign that there is a problem in the operation of the chip, or that the configuration of the chip is not chosen well.

The following exceptions are distinguished:

- Buffer overflow count
- Channel disabled while building message
- Internal error while building message
- Out of sync
- Next grant timeout
- Next request timeout
- New grant but channel empty

Note that in other documents, some of these exceptions are treated as their own kind of message, or as a variant of hit messages or of epoch markers, and that the term *info message* is used instead of *exception*.

Grammar The rules of how the sequence of words (without NOP words), and messages in particular, are composed, can be described in Backus-Naur form as follows:

```
<output> ::= <word-or-message> | <word-or-message> <output>

<word-or-message> ::= <word> | <message>

<word> ::= SOM | TSW | RDA | CON

<message> ::= <hit-message> | <epoch-marker> | <exception>

<exception> ::= <buffer-overflow-count> | <channel-disabled>
                | <message-build-error> | <out-of-sync>
                | <next-grant-timeout> | <next-request-timeout>
                | <new-grant-but-empty>

<hit-message> ::= SOM TSW [<raw-data>] EOM
<raw-data> ::= RDA [<raw-data-continued>]
<raw-data-continued> ::= CON | CON <raw-data-continued>

<epoch-marker> ::= SOM EPM
```

```
<buffer-overflow-count> ::= SOM TSW BOM

<out-of-sync> ::= SOM SYN

<channel-disabled> ::= DIS
<message-build-error> ::= MSB
<next-grant-timeout> ::= NGT
<nest-request-timeout> ::= NRT
<new-grant-but-empty> ::= NBE
```

Notes:

- The `EXD` word type is not used. It was foreseen to be used for sending feature extracted data, which is not implemented at the moment.
- Using `<word-or-message>` in `<output>`, and not simply `<message>`, accounts for unfinished messages being interrupted by an exception.

All word types that can occur as the last word of a message *cannot* occur *before* the end of a message. This means that the end of a message can be determined from a given sequence of words purely by considering the word types, and it is not necessary to take the contents of the message into account, or look ahead of the current word. The current implementation of the "channel switch", which multiplexes the messages from individual channels into a single output sequence, relies on this property.

1.2. Review of message types

1.2.1. Hit messages

Hit messages are generated by a channel when it is triggered. They contain between 0 and 32 ADC samples (9 bits in size) as the payload of the `RDA` word and zero or more `CON` words. The number of samples is determined by the *selection mask* setting and can be less in case of a *multi hit*. The message length (number of words) depends on the number of samples which makes it necessary to detect the last word of a message by inspecting the word types (if the message length were fixed, counting the words would be another option).

In addition, a hit message contains the following metadata fields (with size and comments):

Group ID 8 bits (256 possible values), maybe was necessary when using CBMnet, can be removed in SPADIC 2.1.

Channel ID 4 bits (16 possible values), necessary to distinguish the 16 channels in one group that send messages over the same uplink.

Timestamp 12 bits, the size can be reconsidered for SPADIC 2.1. For every bit removed, the epoch marker rate doubles.

Number of samples 6 bits (64 possible values), necessary to make the last `CON` word unambiguous in case the message did not end normally and possibly contains fewer samples than expected from the selection mask (between 0 and 32 inclusive, hence the choice of 6, not 5 bits).

Actually only the number of samples *in the last `CON` word* need to be distinguished, not the total number of samples in the message, this would save a few bits.

Hit type 2 bits (4 possible values), explains why the message was *generated*. The following cases are distinguished:

- global trigger (DLM in SPADIC 1.x, corresponding command in SPADIC 2.0)
- self trigger
- neighbor trigger
- self and neighbor trigger at the same time

Stop type 3 bits (8 possible values), explains why the message has *ended*. The following (6) cases are distinguished:

- normal end of message
- channel buffer became full while building the message
- ordering FIFO became full while building the message
- multi hit (channel triggered again while building the message)
- channel buffer full and multi hit at the same time
- ordering FIFO full and multi hit at the same time

Comments:

Knowing that the buffer became full while a message was built is not helpful when doing data analysis, but it is necessary to properly end a message when the buffer becomes full, because the channel multiplexing logic requires that the end of the message can be detected. A more appropriate way to do this would be to end the message using a dedicated exception word type instead of a metadata field in the regular hit message.

Likewise for knowing that the ordering FIFO became full while building the message. An entry should be made in the ordering FIFO at the same time the message building starts, so either it's not full and there is no problem, or it's full and no message can be generated at all.

Knowing that there was a multi hit only affects the analysis of the *following* hit message (due to potential pile-up), so this information should more appropriately be stored there.

In summary, apart from informing about a multi hit, the stop type seems to not help in correctly a message, and can probably be replaced by a simple *multi-hit* flag and a new *buffer full* exception type.

Total metadata size

- 8-bit group ID
- 4-bit channel ID
- 12-bit timestamp
- 6-bit number of samples
- 2-bit hit type
- 3-bit stop type

Summed up: 35 bits.

1.2.2. Epoch markers

In order to specify the time when a hit occured, each hit message contains a timestamp, which is incremented at the same rate as the ADC samples are generated. Since the timestamp has a finite length (t bits, where $t = 12$ for all existing SPADIC versions), it periodically overflows (every $2t$ clock cycles), which would make reconstructing the absolute time of a hit from the timestamp ambiguous.

In order to resolve this ambiguity, an *epoch marker* is generated when the timestamp overflows. It is ensured that the epoch marker is inserted into the sequence of messages *before* any hit message with a wrapped-around timestamp.

In principle, the existence of the epoch markers alone would be sufficient to reconstruct the absolute hit times (provided they are counted correctly by the receiving side), but to give some redundancy, and because there is payload available in

the EPM word at no cost, the epoch marker also contains an *epoch count*, which happens to be 12 bits in size as well and is incremented once per epoch marker.

1.2.3. Exception types

Buffer overflow count This is sent after the output buffer of a channel has been full and informs about the number of hits that were missed during that time.

Channel disabled while building message This should not be interesting for the user (who has just intentionally disabled a channel), however this exception type can be necessary for the message multiplexing in a channel group to handle the situation gracefully.

Internal error while building message This should not happen unless there is a programming error in the message building logic, or the internal state of the message builder has become corrupted.

Out of sync This was sent by SPADIC 1.0 and 1.1 when DLM1 was not sent periodically (sending DLM1 at the correct time was required for generating epoch markers).

Since SPADIC 2.0, epoch markers are generated automatically and do not require sending DLM1 (or an equivalent replacement in the STS-XYTER protocol), therefore the "out of sync" exception is not used anymore.

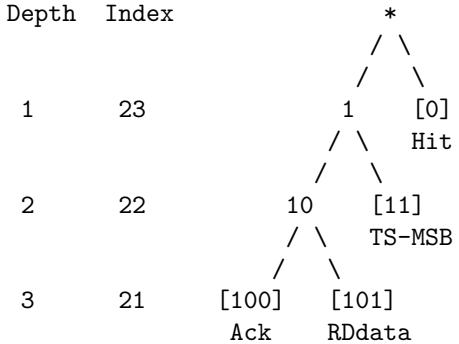
Next grant timeout/Next request timeout/New grant but channel empty These types of exceptions are generated if the internal state of the output multiplexing logic of a channel group has become corrupted.

2. Review of STS-XYTER data format

2.1. Uplink frames

The data output of the STS-XYTER ASIC consists of a sequence of (uplink) *frames* which are 24 bits in size. There are different *frame types*, and like the word types in the SPADIC data format, they are distinguished by their prefix.

The corresponding prefix tree looks as follows:



Unlike in the SPADIC data format, there are no compound data structures (like SPADIC messages) that consist of more than one frame.

2.1.1. Frame types

The following frame types are defined:

Hit This is the frame type used to transmit hit data recorded by individual STS-XYTER channels. In this regard, it is analogous to the SPADIC hit message. Like SPADIC hit messages, it contains a timestamp.

TS-MSB This is sent each time the local timestamp wraps around, in order to make the timestamp contained in hit frames unambiguous. It is therefore analogous to the epoch markers in the SPADIC data format.

Ack This is used to acknowledge control requests (except register read requests).

RDdata This is used to transmit the contents of configuration registers in response to a read request.

2.1.2. Dummy hits

There is one special case of a hit frame, which is called a *dummy hit*. It is a hit frame where all 23 payload bits are zero (since the prefix is a single zero bit, too, the dummy hit is the frame where every bit is zero).

Dummy hits are inserted into the sequence of uplink frames when there is no other data to be sent, because the protocol requires that the frames must be transmitted at a fixed rate without gaps. They can be ignored by the receiving side. This makes the dummy hits analogous to the `NOP` word type in the SPADIC data format.

It is possible to define dummy hits this way, because due to the kind of data contained in a hit frame, a regular hit frame where all payload bits are zero cannot occur.

3. Application of STS-XYTER data format in SPADIC 2.0

In SPADIC version 2.0, the STS-XYTER protocol is used for communicating with the chip and transmitting data, instead of CBMnet, which was used in versions 1.0 and 1.1.

3.1. SPADIC words in hit frames

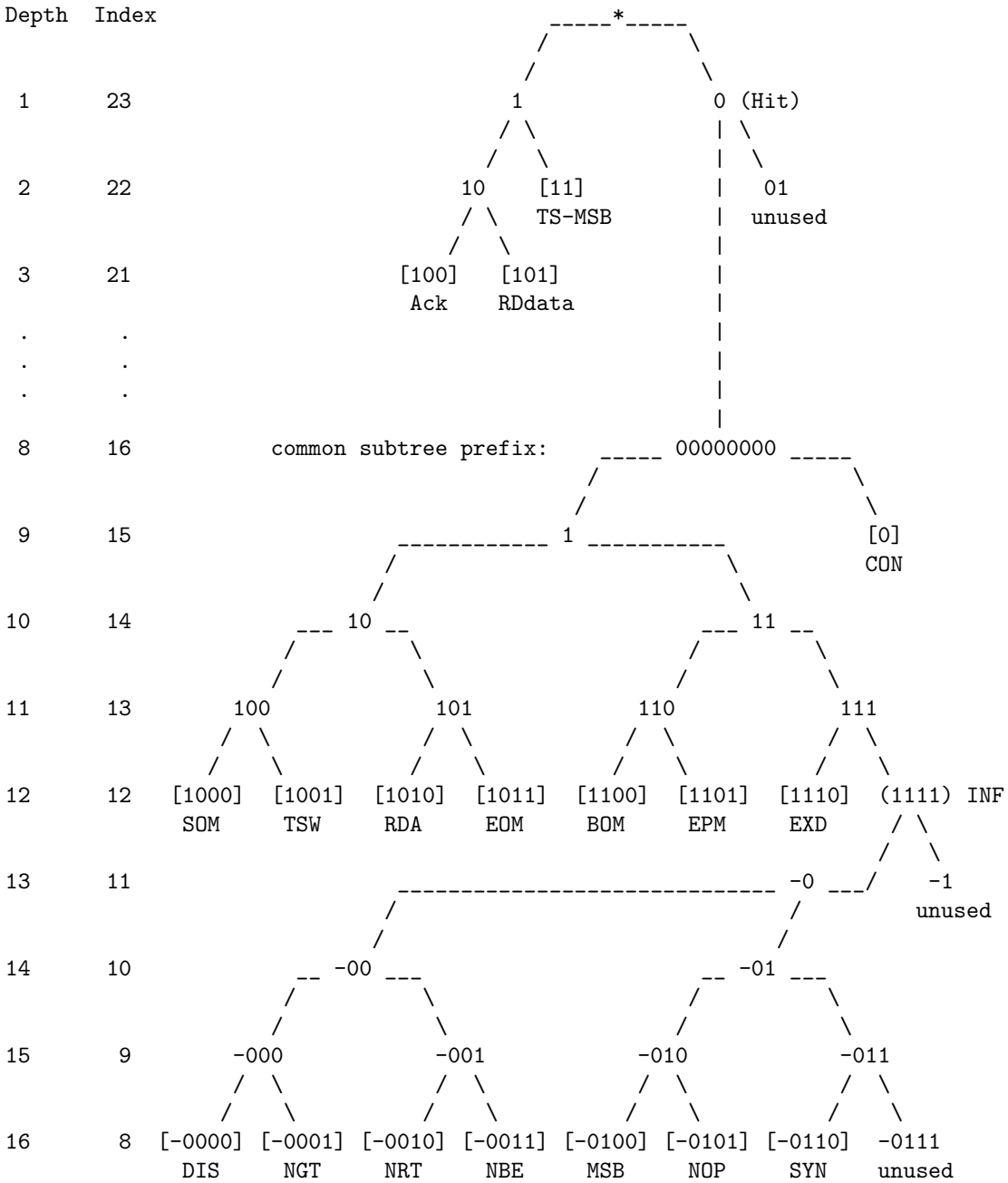
Concerning the SPADIC messages (made from 16-bit words), this is done by simply using one SPADIC word as the payload of one STS-XYTER hit frame and padding the word with 7 zero bits in order to fill up the 23-bit payload.

Redefining the purpose of this frame type is possible because the original interpretation of its contents is only meaningful when actually using the STS-XYTER chip.

The *TS-MSB* frame type is not used in SPADIC 2.0 because the same function is already fulfilled by the epoch markers in the SPADIC data format.

3.1.1. Combined prefix tree

Using SPADIC words as payload of hit frames can also be seen as extending the STS-XYTER data format by replacing the *Hit frame* leaf node in the tree of frame prefixes by the whole tree of word prefixes in the SPADIC format, lowered by 7 levels representing the padding:



This can be interpreted as prepending `00000000` to all prefixes of the SPADIC word types.

3.1.2. Shortcomings

This way of integrating the SPADIC data format in the STS-XYTER protocol has a few shortcomings:

Unused bandwidth Obviously, using only 16 out of 23 bits available as payload of a hit frame is very inefficient.

Dummy frame ambiguity The premise for using a special case of the hit frame as *dummy frame* (which is allowed to be ignored by the receiving side) is not true anymore in the SPADIC case. A hit frame where all payload bits are zero could also be a `CON` word which contains a portion of a recorded pulse where a few ADC samples have the value zero.

This can be easily resolved by ensuring that there is no prefix where every bit is zero.

4. Data format for SPADIC 2.1

4.1. Goals

A new (modified) data format for SPADIC 2.1 should reach the following goals:

Existing properties The following properties of the data format should be kept, in order to not complicate the message handling logic in the chip (channel multiplexing) and in downstream devices:

- Ability to determine the word type from its prefix without context
- Ability to determine the end of a message from the word types

Changes to message contents As explained above (section 1.2.1), there are potential improvements to the contents of transmitted messages which should be addressed:

- Unnecessary metadata can be removed
- Fewer word types can be used to build the messages

Changed word types Some potential savings in transmitted data lie in finding a better set of word types. It should be optimized for the regular case (hit messages and epoch markers), accepting possibly more data to be sent in exceptional cases.

- Unnecessary word types can be removed
- Prefix lengths should be chosen as short as possible
- Shorter prefixes should be chosen for more freqently used word types

This can be expressed as manipulating the tree of prefixes, with the fundamental rules:

- Each word type corresponds to a leaf node
- One leaf node at depth d can be traded for two leaf nodes at depth $d + 1$

One constraint is that *dummy frames* must be unambiguous, which means that prefixes that consist only of `0` bits are not allowed.

4.2. Proposed improvements

4.2.1. Metadata in hit messages

The following changes are proposed (or not) to the metadata contained in hit messages:

- No group ID (8 bits saved)
- Channel ID (4 bits) unchanged
- Timestamp changed to t bits ($12 - t$ bits saved)
- Number of samples indicator changed to s bits ($6 - s$ bits saved)

- Hit type (2 bits) unchanged
- Stop type (3 bits) replaced by “multi hit” flag (2 bits saved), “buffer full” stop reason replaced by new exception type

Notes:

- The number of samples indicator must be stored in the last word of the message, as it is generally not known before.
- The stop type used to be stored in the last word for the same reason. In contrast, this restriction does not apply to the multi hit flag.
- The multi hit flag indicates whether the *current* message was triggered while the previous message was being built (in existing SPADIC versions, the *previous* message would have the stop type “multi hit” and there would be no indication of this fact in the current message).

Total metadata size: $7 + t + s$ bits ($28 - t - s$ bits saved)

The actual values of t and s are determined later:

- t can be chosen relatively freely.
- s must be just large enough to distinguish different numbers of samples resulting in the *same* message length (number of words).

4.2.2. Word types

Hit messages Instead of 5 different word types (`SOM`, `TSW`, `RDA`, `CON`, `EOM`), hit messages are now built from only 3 different word types:

- The payload size of the STS-XYTER frames should be large enough to include the timestamp in the `SOM` word and not have a separate `TSW` word type. If not, t can be made smaller, or metadata can be moved from the `SOM` to the `EOM` word.
- There is no need to distinguish between the first word and the remaining words containing ADC samples, so one of the `RDA` and `CON` word types can be removed. The resulting word type should have the shortest possible prefix, as it is used most frequently (this was already the case for `CON`).

Epoch markers There is already the *TS-MSB* frame type defined in the STS-XYTER protocol, which serves the same purpose as the epoch markers. Therefore, they should be used and the `EPM` word type is not necessary.

Exceptions The *out-of-sync* exception has not been used since SPADIC 2.0. Therefore the `SYN` word type can be removed.

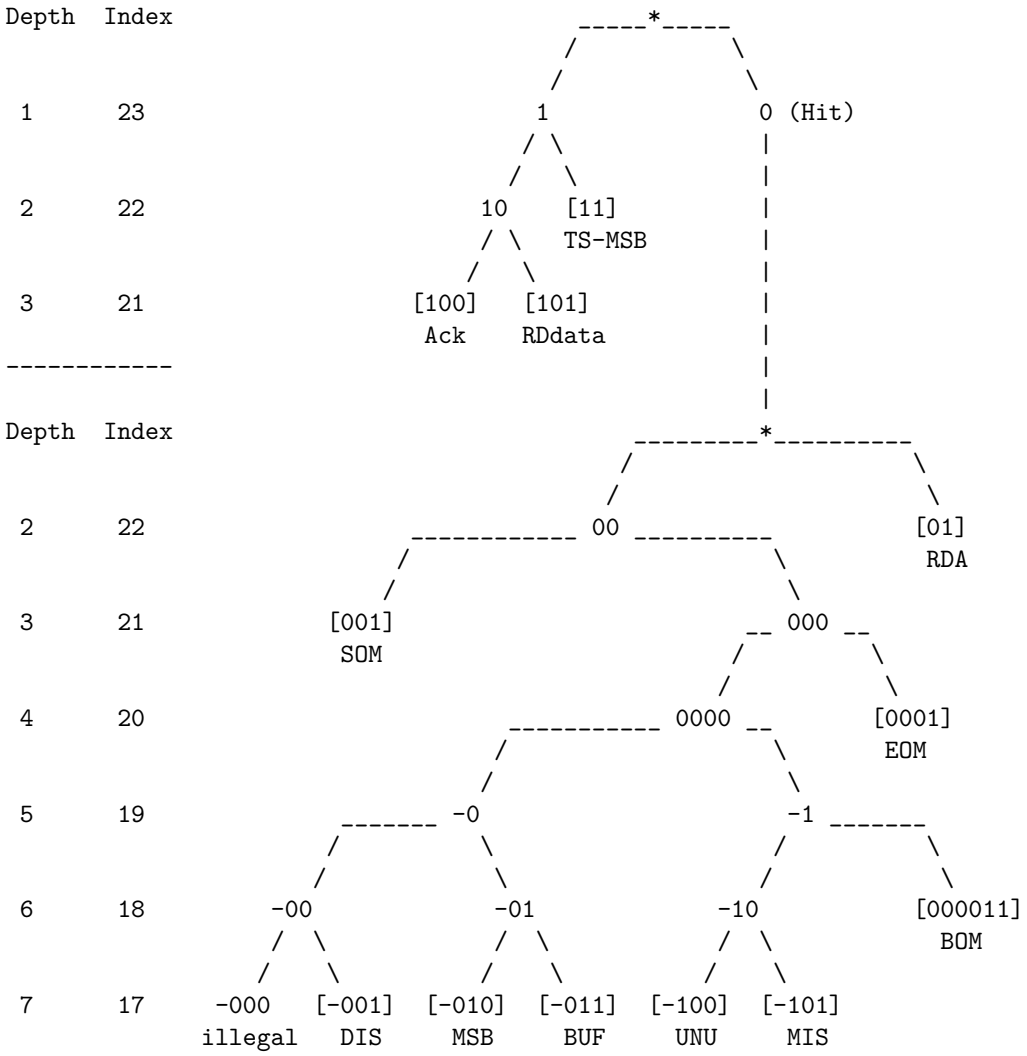
The `NOP` word type can be removed, because its role is already fulfilled by the dummy frame.

The *buffer overflow count* exception consisted of three words: `SOM`, `TSW`, `BOM`, and contained an 8-bit group ID, the 4-bit channel ID, the 12-bit timestamp, and an 8-bit number of hits lost because the channel buffer was full. The group ID is not necessary, the timestamp is also not necessary, because the information contained in this kind of exception is not related to a specific point in time. In the new `SOM` word type (see below) there are 18 payload bits, therefore one `SOM` word is sufficient to transport the channel ID and the number of lost hits which can be increased to 14 bits.

All other exceptions are unchanged and their respective word types are kept. This is no problem as by definition they are only sent in exceptional situations and the size of the STS-XYTER uplink frames allows prefix lengths to provide more than enough different word types.

One *new* exception type is the *buffer full* exception (consisting of a `BUF` word) which replaces the information carried by the "buffer full" stop types.

4.2.3. Resulting prefix tree



4.2.4. Choice of number of samples indicator length

A hit message consists of one `SOM` word, zero or more `RDA` words, and one `EOM` word. Besides the metadata it contains a number of 9-bit ADC samples, spread across the `RDA` words and the available space in the `SOM` and `EOM` words.

Given the length m of the message (in words), the number of bits taken by the word prefixes is:

$P = 3$ (`SOM`) $+ 4$ (`EOM`) $+ (m-2) \cdot 2$ (`RDA`).

The number of bits taken by the metadata is, irrespective of m:

$M = 4$ (channel ID) $+ t$ (timestamp) $+ s$ (number of samples) $+ 2$ (hit type) $+ 1$ (multi hit flag).

This leaves the following number of bits for ADC samples:

$$\begin{aligned} m \cdot 24 - P - M &= m \cdot 24 - 7 - (m-2) \cdot 2 - 7 - t - s \\ &= m \cdot 22 - 10 - t - s. \end{aligned}$$

Considering how many *more* bits for ADC samples there are, compared to the next shortest message, gives:

- $34 - t - s$ for $m = 2$ (compared to no message at all)

- 22 for $m > 2$ (compared to the message with one less `RDA` word)

This value is always less than 36 (but not less than 18), which means that by incrementing the message length, the number of 9-bit samples that fit into the message can grow by no more than 3 (but not by less than 2).

Conclusion Provided that, for a given number of samples S, the shortest possible message length is used, 2 or 3 different numbers of samples result in the same message length. Therefore, $s = 2$ bits are required (and sufficient) to disambiguate the possible numbers of samples contained in a message.

A sensible way to do so is to encode how many ADC samples *less* there are in a message compared to the maximum number of samples that would fit in a message of the same length.

Concern: What happens if words are lost?

- If the end of one message and the start of the following message were lost, two separate messages would be misinterpreted as one message (which could be shorter, longer, or have the same length as either of the original messages).

- If the middle of a message were lost, the message would be misinterpreted as a shorter message.

In the original data format, the 6-bit number of samples indicator would potentially allow to detect this, if the total number of samples reported in the metadata would not match the number of samples found in the interpreted message.

Using the proposed way of indicating the number of samples (difference to maximum number given the message size) would not allow to detect this anymore. It can be used only if it is guaranteed that no words can be lost in a way that results in a valid message which will then be misinterpreted. → **to do** (See also, after reading the next section: Appendix)

4.2.5. Choice of timestamp length

Fundamental considerations In order to find a good value for the timestamp length t, the scenario can be considered where only hit messages, and the mandatory epoch markers in between, are sent, and asking how much of the total amount of data corresponds to the timestamps and epoch markers.

For example, given a hit message length of $m = 5$ words and a timestamp length of $t = 10$ bits,

- One epoch marker must be sent every $2^t = 1024$ clock cycles.

  - An epoch marker consists of one *TS-MSB* frame.
  - One frame consists of 3 bytes.
  - 2 bytes are transmitted per clock cycle.

  The fraction of epoch markers in relation to the total amount of data sent is therefore $E = 1/1024 \cdot 3/2 \approx 0.15\,\%$.

- Of the remaining fraction, 10 bits out of the $m \cdot 24 = 120$ bits of a hit message are the timestamp, i. e., the fraction of timestamps in the total amount of data is $T = (1 - E) \cdot 10/120 \approx 8.32\,\%$.

The combined fraction of timestamps and epoch markers in the total data is therefore $E + T \approx 8.47\,\%$.

Doing this for different timestamp lengths t gives the following results (E, T in percent):

t	E	T	E + T
1	75.00	0.21	75.21
2	37.50	1.04	38.54
3	18.75	2.03	20.78
4	9.38	3.02	12.40
5	4.69	3.97	8.66
6	2.34	4.88	7.23
7	1.17	5.76	6.94
8	0.59	6.63	7.21
9	0.29	7.48	7.77
10	0.15	8.32	8.47
11	0.07	9.16	9.23
12	0.04	10.00	10.03
13	0.02	10.83	10.85
14	0.01	11.67	11.67
15	0.00	12.50	12.50
16	0.00	13.33	13.34

Because E decreases exponentially and T increases somewhat linearly when t becomes larger, there is a minimum value for $E + T$, which in this example is at $t = 7$.

Doing this analysis for different message lengths m gives the following minimum values for $E + T$ and the timestamp lengths t at which they occur:

m	t	E + T
2	6	14.55
3	6	10.48
4	7	8.38
5	7	6.94
6	7	5.98
7	7	5.29
8	8	4.73
9	8	4.27
10	8	3.90
11	8	3.60
12	8	3.35
13	8	3.14
14	8	2.95
15	9	2.79

For shorter messages, the optimum timestamp length is shorter as well, because the timestamp makes up a relatively higher proportion of the message, giving more weight to the benefit from shorter timestamps compared to the increased epoch marker rate.

Wasted bits Considering again the example message length of 5 words (120 bits in total) and the timestamp length of 7 bits, that has so far been determined as optimal for this particular message length:

- 3 (`SOM`) + 6 ($3 \times$ `RDA`) + 4 (`EOM`) = 13 bits are the word prefixes,
- 4 (channel ID) + 2 (hit type) + 1 (multi hit flag) + 2 (number of samples indicator) + t = 16 bits are metadata,

leaving 91 bits for the 9-bit ADC samples, of which one bit is always unused, even if the maximum number of samples (10) is contained in the message.

Therefore, the timestamp could as well be one bit longer without any penalty, which means that the above definition of "optimum timestamp length" does not necessarily give the best results.

In order to correct this, the fraction W of "wasted bits" (which depends on the timestamp length t and the message length m) in the total data must be taken into account in addition to E and T.

In this example, $W = (1 - E) \cdot 1/120 \approx 0.82\,\%$.

Computing the minimum value of $E + T + W$ (with respect to timestamp length t) for a given message length m results in optimum timestamp lengths between 4 and 14 bits for m between 2 and 15 (which is enough to contain 32 samples).

In order to choose a timestamp length which has overall good results nonetheless, the following table is helpful. It shows for each message length m (one row)

- the timestamp length t_{opt} for which $E + T + W$ is minimal,
- the maximum number of samples S in a message of length m, assuming the timestamp length t_{opt} is used (can be one sample more or less for other timestamp lengths),
- the value of $E + T + W$ (in percent) at $t = t_{\text{opt}}$
- for various timestamp lengths the *difference* of $E + T + W$ at that timestamp length compared to the minimum value.

			E+T+W	E+T+W(t) - E+T+W(t_opt)									
m	S	t_opt	t=t_opt	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14
2	3	5	14.62	+4.20	+0.00	+16.21	+15.38	+14.97	+14.76	+14.65	+14.60	+14.58	+14.56	+14.56
3	5	9	12.76	+7.95	+3.85	+1.79	+0.77	+0.26	+0.00	+12.35	+12.30	+12.27	+12.26	+12.25
4	8	4	13.15	+0.00	+4.44	+2.42	+1.40	+0.90	+0.64	+0.52	+0.45	+0.42	+0.41	+9.77
5	10	8	7.21	+8.20	+3.83	+1.64	+0.55	+0.00	+7.20	+7.08	+7.02	+6.98	+6.97	+6.96
6	12	12	8.37	+8.56	+4.26	+2.11	+1.04	+0.50	+0.23	+0.10	+0.03	+0.00	+6.23	+6.22
7	15	7	5.29	+7.86	+3.37	+1.12	+0.00	+4.76	+4.50	+4.37	+4.30	+4.27	+4.25	+4.24
8	17	11	5.80	+8.77	+4.35	+2.14	+1.04	+0.48	+0.21	+0.07	+0.00	+4.65	+4.63	+4.63
9	20	6	5.06	+6.84	+2.28	+0.00	+2.98	+2.43	+2.16	+2.02	+1.96	+1.92	+1.91	+1.90
10	22	10	4.31	+8.84	+4.35	+2.11	+0.98	+0.42	+0.14	+0.00	+3.68	+3.64	+3.63	+3.62
11	24	14	5.31	+5.78	+1.18	+2.21	+1.10	+0.55	+0.27	+0.13	+0.06	+0.03	+0.01	+0.00
12	27	9	3.41	+8.80	+4.26	+1.99	+0.85	+0.28	+0.00	+2.98	+2.91	+2.88	+2.86	+2.85
13	29	13	4.18	+6.35	+4.47	+2.23	+1.11	+0.54	+0.26	+0.12	+0.05	+0.02	+0.00	+2.88
14	32	8	2.95	+8.58	+4.00	+1.72	+0.57	+0.00	+2.38	+2.25	+2.18	+2.14	+2.12	+2.12
15	34	12	3.37	+9.03	+4.50	+2.23	+1.10	+0.53	+0.25	+0.11	+0.04	+0.00	+2.48	+2.47

Conclusions For increasing t (from left to right), when looking at the average values of $E + T + W$ across different message lengths, the fundamental behaviour is still that the value increases for both very short and for longer timestamps (for t between 6 and 10, most of the entries in the table are close to zero).

However, when looking at individual rows, the effect of W becomes apparent, which is that the value steadily decreases, until for a particular timestamp length it increases significantly. This indicates the change between no wasted bits at all at one timestamp length, and the maximum number of samples one less and 8 wasted bits at the next timestamp length.

Because of the numbers involved (22 more bits available for each additional `RDA` word, 9 bits per sample), this change occurs at different timestamp lengths for each message length, and it is practically unavoidable that for every timestamp length there is at least one "unfortunate" message length where the available bits are used relatively inefficiently.

The goal is then to choose a timestamp length where the inefficient message lengths are not the typical use case.

In my opinion:

- The huge penalty at $m = 2$ for $t \geq 6$ is unavoidable because $t < 6$ is generally too inefficient and typically the hit messages should be longer (more than 3 samples).

- For $t > 9$ there is a large penalty at $m = 3$, which might be *the* most frequently used case (4 or 5 samples), so this is out of the question.
- For the remaining timestamp lengths, going from one to the next, my thoughts are:
  - $t = 7$: generally more efficient than $t = 6$, except at $m = 9$, but not very noticeable (+2.98 %).
  - $t = 8$: generally a bit more efficient, but penalty (+4.76 %) at $m = 7$, which might not be relevant (13, 14, or 15 samples).
  - $t = 9$: generally *slightly* more efficient, but large penalty (+7.20 %) at $m = 5$, which might be better avoided (9 or 10 samples). Also minor penalty (+2.38 %) at $m = 14$ (30, 31, or 32 samples).

Personally I would choose between $t = 7$ and $t = 8$.

4.3. Summary

4.3.1. Word/frame types

The defined word types (including the frame types defined in the STS-XYTER protocol), with their prefixes, are:

- `01`.....: `RDA` word
- `11`.....: *TS-MSB* frame
- `100`.....: *Ack* frame
- `101`.....: *RDdata* frame
- `001`.....: `SOM` word
- `0001`.....: `EOM` word
- `000011`.....: `BOM` word
- `0000001`.....: `DIS` word
- `0000010`.....: `MSB` word
- `0000011`.....: `BUF` word
- `00001000`.....: `NGT` word
- `00001001`.....: `NRT` word
- `00001010`.....: `NBE` word
- `000000000000000000000000`: *Dummy* frame

(All word types starting with a `0` bit are "subtypes" of the *Hit* frame type defined in the STS-XYTER protocol.)

(The *TS-MSB* frame type and the `MSB` word type are not to be confused.)

4.3.2. Grammar

The messages contained in the sequence of *Hit* frames, with *Dummy* frames filtered out, and *TS-MSB* frames, are formed according to the following rules:

```
<output> ::= <word-or-message> | <word-or-message> <output>

<word-or-message> ::= <word> | <message>

<word> ::= SOM | RDA

<message> ::= <hit-message> | <epoch-marker> | <exception>

<exception> ::= <buffer-overflow-count> | <channel-disabled> | <buffer-full>
                | <message-build-error> | <next-grant-timeout>
```

```
                | <next-request-timeout> | <new-grant-but-empty>

<hit-message> ::= SOM [<raw-data>] EOM
<raw-data> ::= RDA | RDA <raw-data>

<epoch-marker> ::= TS-MSB

<buffer-overflow-count> ::= BOM
<channel-disabled> ::= DIS
<buffer-full> ::= BUF
<message-build-error> ::= MSB
<next-grant-timeout> ::= NGT
<nest-request-timeout> ::= NRT
<new-grant-but-empty> ::= NBE
```

Note: Using `<word-or-message>` in `<output>`, and not simply `<message>`, accounts for unfinished messages being interrupted by an exception.

4.3.3. Message format

The different message and exception types are formatted as follows:

Hit messages Consist of one `SOM` word, zero or more `RDA` words, and one `EOM` word.

The `SOM` word contains

- the 4-bit channel ID `cccc`,
- the t-bit timestamp `ttttttt`,
- the multi-hit flag `m`
- the 2-bit hit type `hh`.

The `EOM` word contains

- the 2-bit number of samples indicator `nn`,
- possibly (depending on message length and timestamp length) a number of unused bits (`.`).

The remaining space in all words of the message is filled with ADC samples (`>--------`).

Example message with 3 `RDA` words, assuming $t = 7$:

```
Word type  Format

SOM        001ccccttttttmhh>------
RDA        01-->-------->-------->-
RDA        01------->-------->-----
RDA        01--->-------->-------->
EOM        0001nn-------->--------.
```

`nn` is the number of samples that would *additionally* fit in a message of the same size.

The example message can contain 8 (`nn` = 2), 9 (`nn` = 1), or 10 (`nn` = 0) samples. A message with 7 samples would be only 4 words long.

Indicating the number of samples this way (as opposed to a 6-bit number of total samples contained) is possible, provided that

- the shortest possible message length is used for a given number of samples,
- it is guaranteed that no part of a message can be lost in a way which would result in a different valid hit message.

Exception types related to hit messages:

- If a channel is disabled while a message is built, a *channel disabled* exception consisting of a `DIS` word is sent, which contains the 4-bit channel ID and 13 unused bits:

  `0000001.............cccc`

- If the output buffer becomes full while a message is built, a *buffer full* exception consisting of a `BUF` word is sent, which contains the 4-bit channel ID `cccc`, the 2-bit *buffer full* status code `bb`, and 11 unused bits:

  `0000011bb...........cccc`

  The status codes are:

  - `01`: channel buffer full
  - `11`: channel buffer full and multi hit
  - `10`: multi hit, but ordering FIFO full

- If the internal state of the message building logic is corrupted, a *message build error* exception consisting of an `MSB` word is sent, which also contains the channel ID and 13 unused bits:

  `0000010.............cccc`

Epoch markers Epoch markers consist of one *TS-MSB* frame, which contains the 6-bit epoch count `Eeeeee`, replicated 3 times, and a 4-bit CRC checksum `CCCC`, according to the specification of the STS-XYTER protocol:

`11EeeeeeEeeeeeEeeeeeCCCC`

Exception types

- *Channel disabled*, *Message build error*, *Buffer full*: Already described above
- *Buffer overflow message*: Consists of a `BOM` word and contains the 4-bit channel ID `cccc` and the 14-bit number of hits lost due to the channel buffer being full:

  `000011nnnnnnnnnnnnnnnncccc`

- *Next grant timeout*/*Next request timeout*/*New grant but channel empty*: They consist of an `NGT`, an `NRT`, and an `NBE` word, respectively. `NGT` and `NBE` contain the 4-bit channel ID, `NRT` does not because it indicates an error that is not specific to a channel.
  - `NGT`: `00001000.............cccc`
  - `NRT`: `00001001.................`
  - `NBE`: `00001010.............cccc`

A. Fallback to larger number of samples indicator

In case we must go back to $s = 6$ bits to indicate the number of samples, here is the alternative "timestamp length efficiency" table for that case:

		E+T+W	E+T+W(t) - E+T+W(t_opt)										
m	S	t_opt	t=t_opt	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14
2	2	10	20.95	+7.31	+3.59	+1.74	+0.81	+0.35	+0.12	+0.00	+18.68	+18.66	+18.65	+18.64
3	5	5	11.31	+4.36	+0.00	+10.03	+9.08	+8.61	+8.37	+8.26	+8.20	+8.17	+8.15	+8.15
4	7	9	9.64	+8.23	+3.98	+1.86	+0.80	+0.27	+0.00	+9.23	+9.17	+9.14	+9.12	+9.12
5	9	13	10.85	+1.55	+4.16	+2.07	+1.03	+0.51	+0.24	+0.11	+0.05	+0.02	+0.00	+7.49
6	12	8	6.11	+8.30	+3.87	+1.66	+0.55	+0.00	+5.95	+5.83	+5.76	+5.73	+5.71	+5.70
7	14	12	7.18	+8.67	+4.32	+2.14	+1.05	+0.51	+0.24	+0.10	+0.03	+0.00	+5.34	+5.33
8	17	7	4.77	+7.90	+3.39	+1.13	+0.00	+4.10	+3.83	+3.69	+3.63	+3.59	+3.58	+3.57
9	19	11	5.16	+8.83	+4.38	+2.15	+1.04	+0.49	+0.21	+0.07	+0.00	+4.13	+4.11	+4.11
10	22	6	4.79	+6.86	+2.29	+0.00	+2.56	+2.01	+1.74	+1.60	+1.53	+1.50	+1.48	+1.47
11	24	10	3.93	+8.88	+4.37	+2.11	+0.99	+0.42	+0.14	+0.00	+3.34	+3.30	+3.29	+3.28
12	26	14	4.87	+6.08	+1.47	+2.22	+1.11	+0.55	+0.27	+0.13	+0.06	+0.03	+0.01	+0.00
13	29	9	3.17	+8.82	+4.27	+1.99	+0.85	+0.28	+0.00	+2.74	+2.67	+2.63	+2.62	+2.61
14	31	13	3.89	+6.57	+4.49	+2.24	+1.11	+0.55	+0.26	+0.12	+0.05	+0.02	+0.00	+2.67
15	34	8	2.80	+8.59	+4.01	+1.72	+0.57	+0.00	+2.21	+2.07	+2.00	+1.96	+1.94	+1.94

And in case 5 bits are used (6 bits are in my opinion only necessary because we distinguish between 0 and 32 samples, which are 33 possible values), for example by excluding a message with 0 samples:

		E+T+W	E+T+W(t) - E+T+W(t_opt)										
m	S	t_opt	t=t_opt	t=4	t=5	t=6	t=7	t=8	t=9	t=10	t=11	t=12	t=13	t=14
2	2	11	22.97	+7.17	+3.56	+1.75	+0.85	+0.40	+0.17	+0.06	+0.00	+18.71	+18.70	+18.70
3	5	6	10.48	+6.45	+2.15	+0.00	+11.28	+10.82	+10.58	+10.47	+10.41	+10.38	+10.37	+10.36
4	7	10	10.55	+8.27	+4.07	+1.97	+0.92	+0.39	+0.13	+0.00	+9.30	+9.27	+9.26	+9.25
5	10	5	8.66	+4.49	+0.00	+5.08	+4.04	+3.53	+3.27	+3.14	+3.07	+3.04	+3.02	+3.02
6	12	9	6.52	+8.51	+4.12	+1.92	+0.82	+0.27	+0.00	+6.10	+6.04	+6.01	+5.99	+5.98
7	14	13	7.75	+3.78	+4.31	+2.15	+1.06	+0.52	+0.25	+0.12	+0.05	+0.02	+0.00	+5.35
8	17	8	4.73	+8.42	+3.93	+1.68	+0.56	+0.00	+4.39	+4.26	+4.19	+4.16	+4.14	+4.13
9	19	12	5.59	+8.82	+4.39	+2.18	+1.07	+0.52	+0.24	+0.10	+0.03	+0.00	+4.15	+4.14
10	22	7	4.05	+7.96	+3.41	+1.14	+0.00	+3.16	+2.89	+2.75	+2.68	+2.65	+2.63	+2.62
11	24	11	4.24	+8.91	+4.42	+2.18	+1.05	+0.49	+0.21	+0.07	+0.00	+3.37	+3.36	+3.35
12	27	6	4.38	+6.88	+2.29	+0.00	+1.94	+1.39	+1.11	+0.97	+0.90	+0.86	+0.85	+0.84
13	29	10	3.35	+8.93	+4.40	+2.13	+0.99	+0.43	+0.14	+0.00	+2.81	+2.78	+2.76	+2.75
14	31	14	4.18	+6.55	+1.93	+2.24	+1.11	+0.55	+0.27	+0.13	+0.06	+0.03	+0.01	+0.00
15	34	9	2.79	+8.85	+4.28	+2.00	+0.86	+0.29	+0.00	+2.35	+2.28	+2.25	+2.23	+2.22

However, I am quite confident that it can be ensured that only complete messages are sent out of the chip, and therefore $s = 2$ would work.