Frame Synchronization without Attached Sync Markers

Jon Hamkins

Jet Propulsion JPL
California Institute of Technology

March 9, 2011

(c) 2011 California Institute of Technology. Government sponsorship acknowledged.
Introduction

Conventional frame synchronization

- Attached Sync Markers (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:

![Attached Sync Marker]

• For CCSDS low-density parity-check (LDPC) codes, the ASM is the 64-bit pattern:

000000110100011101101101100011100100111001010001001010110110000
Introduction

Conventional frame synchronization

- *Attached Sync Markers* (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:

```
... ASM | Codeword | ASM | Codeword | ASM ...
```

- Received symbols are correlated with local copy of ASM
Introduction

Conventional frame synchronization

• *Attached Sync Markers* (ASMs) are inserted between codewords
• ASM+codewords are sent one after the other, without gaps:
Introduction

Conventional frame synchronization

• *Attached Sync Markers* (ASMs) are inserted between codewords
• ASM+codewords are sent one after the other, without gaps:
Introduction

Conventional frame synchronization

- *Attached Sync Markers* (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:
Introduction

Conventional frame synchronization

- *Attached Sync Markers* (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:
Introduction

Conventional frame synchronization

- *Attached Sync Markers* (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:
Introduction

Conventional frame synchronization

- Attached Sync Markers (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:
Introduction

Conventional frame synchronization

- *Attached Sync Markers* (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:
Introduction

Conventional frame synchronization

- *Attached Sync Markers* (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:

```
... ASM  Codeword  ASM  Codeword  ASM  ...
```

When high correlation is found, sync is declared
Introduction

Conventional frame synchronization

- *Attached Sync Markers* (ASMs) are inserted between codewords
- ASM+codewords are sent one after the other, without gaps:

```
... ASM  Codeword  ASM  Codeword  ASM ...
```

- This method has been used successfully for decades for legacy codes
- It was also successfully tested for the emerging LDPC code standards
### Overhead of ASMs

Overhead of including ASM, for CCSDS codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Transmitted ASM Length</th>
<th>Codeword Length</th>
<th>$E_b/N_0$ penalty (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reed-Solomon</td>
<td>32</td>
<td>2040</td>
<td>0.1</td>
</tr>
<tr>
<td>RS+CC</td>
<td>64</td>
<td>&gt;4080</td>
<td>&lt;0.1</td>
</tr>
<tr>
<td>Turbo</td>
<td>192</td>
<td>&gt;3568</td>
<td>&lt;0.1</td>
</tr>
<tr>
<td>LDPC</td>
<td>64</td>
<td>&gt;2048</td>
<td>&lt;0.1</td>
</tr>
<tr>
<td>LDPC</td>
<td>64</td>
<td>≤2048</td>
<td>0.1 to 0.2</td>
</tr>
</tbody>
</table>
Overhead of including ASM, for CCSDS codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Transmitted ASM Length</th>
<th>Codeword Length</th>
<th>$E_b/N_0$ penalty (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reed-Solomon</td>
<td>32</td>
<td>2040</td>
<td>0.1</td>
</tr>
<tr>
<td>RS+CC</td>
<td>64</td>
<td>&gt;4080</td>
<td>&lt;0.1</td>
</tr>
<tr>
<td>Turbo</td>
<td>192</td>
<td>&gt;3568</td>
<td>&lt;0.1</td>
</tr>
<tr>
<td>LDPC</td>
<td>64</td>
<td>&gt;2048</td>
<td>&lt;0.1</td>
</tr>
<tr>
<td>LDPC</td>
<td>64</td>
<td>≤2048</td>
<td>0.1 to 0.2</td>
</tr>
<tr>
<td>BCH</td>
<td>80*</td>
<td>64**</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Eliminating the ASM would significantly increase uplink coding gain.

* 16-bit marker and 64-bit tail sequence
** Assuming 1 ASM per codeword (the minimum acquisition-time configuration)
Clock Distribution

In a hardware implementation, clocks run at the *bit rate* and *symbol rate*:

To avoid buffering, it is helpful if the symbol rate / bit rate is a simple ratio. Here are the ratios for CCSDS standard LDPC codes:

<table>
<thead>
<tr>
<th>Input length (bits)</th>
<th>Rate</th>
<th>ASM length</th>
<th>Symbol rate to bit rate ratio (including ASM)</th>
<th>Symbol rate to bit rate ratio (without ASM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1/2</td>
<td>64</td>
<td>33 : 16</td>
<td>2 : 1</td>
</tr>
<tr>
<td>4096</td>
<td>1/2</td>
<td>64</td>
<td>129 : 64</td>
<td>2 : 1</td>
</tr>
<tr>
<td>16384</td>
<td>1/2</td>
<td>64</td>
<td>513 : 256</td>
<td>2 : 1</td>
</tr>
<tr>
<td>1024</td>
<td>2/3</td>
<td>64</td>
<td>25 : 16</td>
<td>3 : 2</td>
</tr>
<tr>
<td>4096</td>
<td>2/3</td>
<td>64</td>
<td>97 : 64</td>
<td>3 : 2</td>
</tr>
<tr>
<td>16384</td>
<td>2/3</td>
<td>64</td>
<td>385 : 256</td>
<td>3 : 2</td>
</tr>
<tr>
<td>1024</td>
<td>4/5</td>
<td>64</td>
<td>21 : 16</td>
<td>5 : 4</td>
</tr>
<tr>
<td>4096</td>
<td>4/5</td>
<td>64</td>
<td>81 : 64</td>
<td>5 : 4</td>
</tr>
<tr>
<td>16384</td>
<td>4/5</td>
<td>64</td>
<td>321 : 256</td>
<td>5 : 4</td>
</tr>
</tbody>
</table>

*Eliminating the ASM would simplify clock distribution.*
New Frame Sync Approach

- Eliminate ASMs from transmission – just transmit codewords
- Attempt to decode at every possible offset
- When correct decoding results, frame sync has been found
New Frame Sync Approach

- Eliminate ASMs from transmission – just transmit codewords
- Attempt to decode at every possible offset
- When correct decoding results, frame sync has been found
New Frame Sync Approach

- Eliminate ASMs from transmission – just transmit codewords
- Attempt to decode at every possible offset
- When correct decoding results, frame sync has been found
New Frame Sync Approach

- Eliminate ASMs from transmission – just transmit codewords
- Attempt to decode at every possible offset
- When correct decoding results, frame sync has been found
New Frame Sync Approach

- Eliminate ASMs from transmission – just transmit codewords
- Attempt to decode at every possible offset
- When correct decoding results, frame sync has been found
Solution to a False Sync Problem

• Problem: The CCSDS LDPC codes are quasi-cyclic. A cyclic shift by 1 symbol is still decodable (and to a wrong codeword!):

  Correct sync  
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Codeword</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Decoder</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
</tbody>
</table>
  Successfull decoding              

  Sync offset by 1 symbol  
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Codeword</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Decoder</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
</tbody>
</table>
  Successfull decoding (but wrong!)  

• Solution: Use the CCSDS-recommended randomizer at transmitter:

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Codeword</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
</tbody>
</table>
  +                                  
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomized Codeword</td>
</tr>
</tbody>
</table>

  PN sequence

• Randomized codewords
  – Do not have the quasi-cyclic property
  – Do not falsely decode at incorrect offset

• Conclusion: use the randomizer when using the new sync method
Properties of the new approach:

- All ASM transmissions are eliminated
- The otherwise-idle decoder is now utilized during synchronization
- Algorithm is guaranteed to find correct offset whenever decodable data is present
- Up to $n$ offsets must be tested, where $n$ is the codeword length
A Faster Version of the Synchronizer

Attempting full decoding at every offset can take a while. Can we make it faster?

With LDPC and turbo codes:

- Decoding consists of a series of iterations
- After each iteration, each code symbol is assigned a log-likelihood ratio (LLR), relating the probability that the symbol is a 0 or a 1
- A properly synchronized codeword will converge fundamentally differently from an improperly synchronized codeword

![Diagram of decoder iterations]

Idea:
Use only a few decoder iterations and develop a metric to distinguish the correct and incorrect sync states.
Developing a Metric, Using Variable Nodes

Distribution of variable node LLRs, when correctly synchronized:

After 10 iterations, many nodes have high abs LLRs.

Synchronized

Symbol is 1 with high probability
Symbol is uncertain
Symbol is 0 with high probability

10 iterations
5 iterations
2 iterations
0 iterations

Quantized LLR

Probability

10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8

-127 -100 -50 0 50 100 127

r=4/5, k=1024 AR4JA LDPC code
Developing a Metric, Using Variable Nodes

Distribution of variable node LLRs, when incorrectly synchronized:

After 10 iterations few nodes have high abs LLRs

Not Synchronized

Probability

Quantized LLR

200 iterations 10 iterations 5 iterations 2 iterations 0 iterations

Symbol is 1 with high probability

Symbol is uncertain

Symbol is 0 with high probability

$r=4/5$, $k=1024$ AR4JA LDPC code
Developing a Metric, Using Check Nodes

Distribution of *number of satisfied check nodes*, when correctly synchronized:

![Graph showing distribution of satisfied check nodes](image)

- After 10 iterations, most check nodes are satisfied.
- 200 iterations.
- 5 iterations.
- 2 iterations.

$r=4/5, k=1024$ AR4JA LDPC code
Distribution of *number of satisfied check nodes*, when **not synchronized**:

After 10 iterations most check nodes are **not** satisfied

---

Number of failed check nodes

- 2 iterations
- 5 iterations
- 10 iterations
- 200 iterations

Probability

- $10^{-4}$
- $10^{-3}$
- $10^{-2}$
- $10^{-1}$
- $10^{0}$

$r=4/5, k=1024$ AR4JA LDPC code
Possible Metrics

• Metric for variable nodes:

\[ M = \sum_{i=1}^{n} f(\lambda_i) \]

where \( \lambda_i \) is the ith LLR and \( f(.) \) is an even, monotonically increasing function

– Reasonable choices:

1. \( f(x) = |x|^a \), for some real positive \( a \)
2. \( f(x) = e^{|x|} \)
3. \( f(x) = \log(1 + |x|) \)
4. \( f(x) = I_{\{|x| \geq \eta\}} \), where \( I \) is the indicator function and \( \eta \) is a threshold

• Metric for check nodes:

\[ M = \sum_{i=1}^{n-k} I_{\{\text{check node } i \text{ satisfied}\}} \]

i.e., count the number of satisfied check nodes
New Frame Sync Algorithm

1. Initialize \( j = 0 \)
2. Collect \( n \) symbols from channel, starting at symbol \( j \)
3. Soft derandomize
4. Decode for \( I \) iterations
5. Form metric \( M(j) \)
6. Increment \( j \)

If \( j < n \):
- Declare armax \( M(j) \) to be correct offset

At this point, full decoding can occur. If this fails, the sync process may be repeated.

Or, full decoding may be attempted at the offset of the 2\textsuperscript{nd}, 3\textsuperscript{rd}, etc. highest metric.
Frame Sync Performance

Performance is insensitive to choice of metric

Conclusion: pick a simple-to-compute metric (|x|)
Sync performance is not as good as code performance.

**Conclusion:** Adequate sync is not achieved within one codeframe length.
Sync Performance: Based on 2 Codewords

For >10 iterations, sync performance is better than code performance.

Conclusion: Good sync error rate is achieved within two codeframe lengths.
Decoder acquisition time is reasonably good:

- Acquires in ~2 s, for data rates > 10 kbps
- Acquires in ~15 ms, for data rates > 500 kbps

**Acquisition time is faster at higher data rates**

$r=4/5, k=1024$ AR4JA LDPC code
Conclusions

• A new frame synchronizer was presented
  – Eliminates need to transmit attached sync markers (ASMs)
  – In ~10 iterations, decoder can distinguish between sync and non-sync states
• Advantages:
  – 3.5 dB coding gain for standard uplink codes
  – 3 dB coding gain for proposed uplink LDPC codes of similar length
  – 0.2 dB coding gain for CCSDS LDPC codes
  – Simplified clocking in hardware on spacecraft and on ground