HIGH-SPEED SOFTWARE FRAME SYNCHRONIZER USING CIRCULAR BUFFER

In-Hoi KOO[1], Sang-Il Ahn[1], Tae-Hoon Kim[2], and Young-Bo Sakong[2]

Satellite Mission Operation Department, Korea Aerospace Research Institute[1]
freewill@kari.re.kr[1], siahn@kari.re.kr[1]
SOLETOP Inc. Satellite Image Dept. [2]
freekid99@soletop.com[2], ybsakong@soletop.com[2]

ABSTRACT: For a satellite data communication, the technology of frame synchronization is widely used between a sender and a receiver. Last year, we suggested zero-loss frame synchronization [1] using pattern search and using bits threshold search algorithm that is based on SIMD technology [2,3]. This algorithm could solve both of hardware and software drawbacks, which are frame loss and low processing performance. However, this algorithm didn't optimize the processing of output data, synchronized data, which caused overhead to the memory allocation and the memory copy. Consequently, the performance of the frame synchronizer application was degraded.
In this paper, we enhance previous work using a circular buffer in order to optimize the output data processing. The performance comparison with the previous algorithm shows that the enhanced proposed approach dramatically outperforms in the output data processing speed.

KEY WORDS: CCSDS, Frame Synchronization, SSE2, Circular Buffer

1. INTRODUCTION

The communication between satellite and ground station has been changed from analogue communication to digital communication. The CCSDS (Consultative Committee for Space Data Systems) recommended some standards such as [4] for digital communications between satellite and ground station. In current satellite communication, the satellite sends huge data to the ground station. Therefore, the ground station should possess huge data processing capability.

The zero-loss frame synchronization we proposed in [1] solved the problem of slow processing and the problem of data loss is from hardware process. It enhanced the deficiencies of low speed software processing using the SSE2 [2,3] and using the pattern search. It also minimized frame loss with Pending Buffer.

This approach, although effective in terms of speed and frame loss, didn't give an attention to the data processing after frame synchronization.

Thus in this paper, using a circular buffer, we optimize data processing. The enhanced frame synchronizer is described in detailed in the section 4.

2. FRAME SYNCHRONIZATION & SSE2

2.1 Frame Synchronization

The frame synchronization is to align incoming frame in continuous bit stream by indicating the start or stop of frame with a special bit combination.

It has four kinds of states; Search, Check, Lock, Flywheel State. The explanation of each states of Figure 1 is as follows.

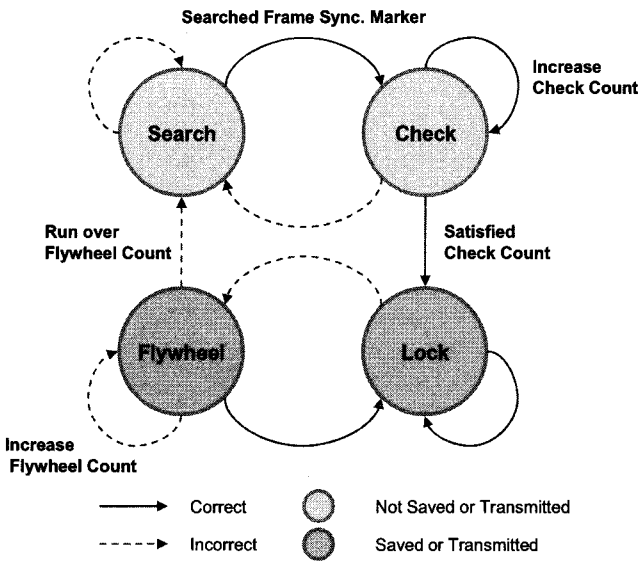


Figure 1 Frame Synchronization Final State Diagram

First of all, in Search State, we look for the Frame Synchronize Marker (hereafter, FSM) in the incoming data. The incoming data is being shifted by unit of a bit until we find the FSM which can be varied according to modulation methods. All possible values are compared with the incoming data and finally we can detect the FSM which does not exceed the bit error threshold. When FSM is successfully searched, the state is transited to Check State.

Check State detects the FSM in the unit of a frame using pre-detected FSM in Search State. When the FSM detection is successful w.r.t. the configured check count number, the state is transited to Lock State. If not, the state goes back to the Search State.

Like Check state, Lock State detects the FSM in the incoming data with the unit of a frame. When FSM is detected, detected frame is saved or transferred to next step. If not, it triggers the start of Flywheel state.

Flywheel State tries the FSM detection with the size of frame. When FSM detection fails, the flywheel count value increases by 1. If the flywheel count value exceeds the predefined value, the state goes back to Search State. When FSM detection is done, the state jumps to Lock State. In Lock State, the frame data is saved or transferred to next processing step.

2.2 SSE2 (Streaming SIMD Extensions2)

The Sinlge-Insturction Multiple-Data (SIMD) can be applied in a loop whose instruction is repeated. The performance of SIMD can be improved if SIMD when the incoming data is arranged in a row like Figure 2: SSE2.

Thanks to the SSE2, just single instruction can be used to the iterative loops with multiple incoming data.

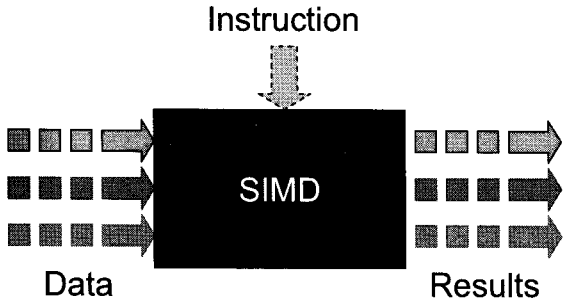


Figure 2 SIMD Processing

3. ASSUMPTIONS IN IMPLEMENTATION

3.1 Bit Pattern

We can classify all possible Bit Pattern into five patterns of Table 1 according to the CCSDS recommendations.

Name	Value
For non-turbo coded data	1ACFFC1D
For rate-1/2 turbo coded data	034776C7272895B0
For rate-1/3 turbo coded data	25D5C0CE8990F6C9461BF79C
For rate-1/4 turbo coded data	034776C7272895B0 FCB88938D8D76A4F
For rate-1/6 turbo coded data	25D5C0CE8990F6C9461BF79C DA2A3F31766F0936B9E40863

Table 1 Frame Synchronization Bit Patterns

In this paper, the Turbo Coding is not applied to the simple 32bit FSM value of "1ACFFC1D". Figure 3 shows bit pattern in FSM without Turbo Coding.

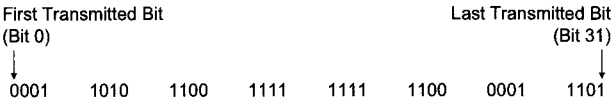


Figure 3 FSM for non-turbo coded data

3.2 Modulation/Demodulation

It is Quadrature Phase Shift Keying (QPSK) that is widely used modulation scheme of data transmission in existing satellites such as MTSAT-1R, MSG, METOP, and KOMPSAT. The QPSK, one of Phase Shift Keying (PSK), uses 4 phase values to express the 2bits information; 0°, 90°, 180°, 270°.

4. PERFORMANCE ENHANCEMENT

4.1 Pending Buffer

The concept of Pending Buffer is to prevent any data loss when new data is arrived before the FSM Search State is not completed on the previous data.

In Figure 4, Input Data#1 is current data under processing and Input Data#2 is next data to be processed. When Search State frame and Check State frame are in Input Data#1, two frames in the red box are copied to the Pending Buffer and then FSM search is continued.

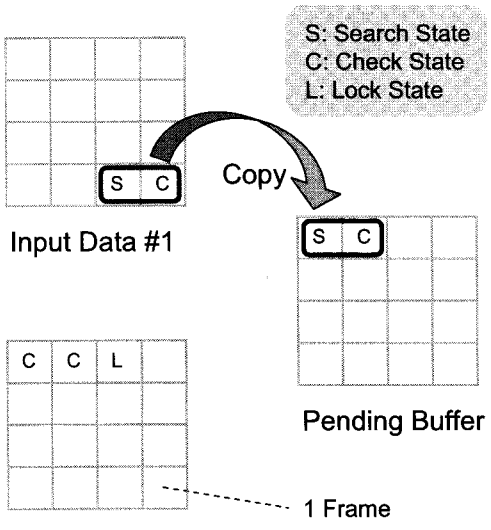


Figure 4 Pending Buffer

4.2 Pattern Search

Pattern Search is to check whether identical bits stream is repeated within the incoming data. When the satellite does not transmit any data, the pattern of incoming data from receiver to serial telemetry card is "00" or "FF." In this case, it is not necessary to find FSM in Search State.

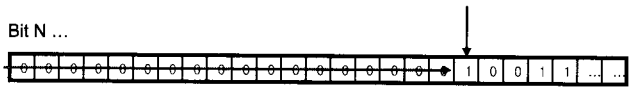


Figure 5 Pattern Search

Pattern Search saves the first 4byte data and compares them with next 4byte data after 1bit shift. If these two values are same, the frame synchronizer assumes no data is coming from the satellite.

As shown in Figure 5, Search State can easily move to the value of "1" using Pattern Search. This pattern search concept is very useful when no data transmission or data loss situation.

4.3 Bits Threshold with SSE2

In Search State, it's FSM when the nearest Hamming distance vathfwlrlue under Bits threshold for 4 possible FSM in QPSK is FSM. In case of no FSM, 1bit shift operation and hamming distance calculation continue until successful FSM search. These two activities of bit shift and hamming distance calculation require huge computational load.

SSE2 was applied to quick comparison between input data and 4 possible FSM values. With SSE2 technology, simple one instruction command lead equivalent effects obtained 4 times individual commands.

Consequently, FSM can be detected by comparing the bit threshold value and FSM Hamming distance after calculating.

To apply the SSE2 in bit threshold comparison sequence, all 4 possible FSM value of "1ACFFC1D" were calculated in advance and saved in XMM1 register and 4 bytes of data in input data buffer was saved in XMM2 register up to 4 times and finally Hamming distance calculation between these two XMM were calculated.

When the Hamming distance is less than the Bits threshold, we can see FSM detection was successful. But when there is no Hamming distance less than threshold, the 1bit shift operation is done and the new Hamming distance calculation and comparison sequence is started until successful FSM detection.

Using 128-bit XMM register, 4 of phase values are compared with just 1 instruction command and simply we can expect 4-times higher processing power from calculation load's point a view.

Figure 6 shows an example of bits threshold calculation using SSE2.

4-byte value of Input Data Buffer is 0x4F9AA948. XMM1 includes 4 kinds of FSM for different phases. XMM2 includes 4 times 4-byte data of 0x4F9AA948. XMM1 and XMM2 are just exclusive OR-ed and its results are recorded in XMM0 and its hamming distance was calculated. If value of 2 is configured to bits threshold, the 4-byte of FSM showing its Hamming distance under value of 2 is FSM. In Figure 6, phase is 90° and FSM value is 0x4F9AA948.

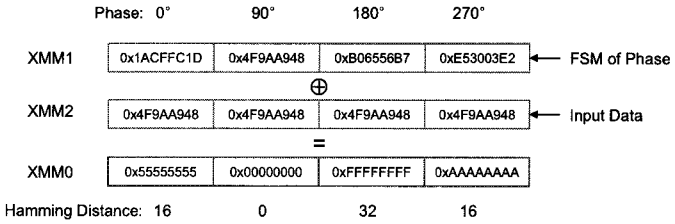


Figure 6 Bits Threshold using SSE2

5. OUT BUFFER HANDLING

5.1 Linked List

The process of Frame Synchronization had a Linked List for handling Out buffer as showing figure 7.

Lined List is possible to assign the memory and dynamic assignment of memory so it is easy to delete the date, insert the data because it is able to use the memory efficiently.

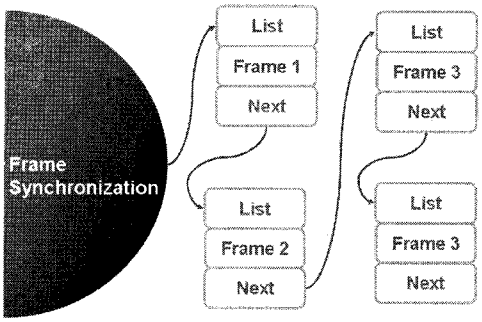


Figure 7 Linked List structure

Sometimes Linked List makes lowering of capacity because of Overhead which is caused by excessive assigning memory or cancellations in a satellite system.

The defects are that the algorism is complicated and storing place is not enough for keeping the pointer.

5.2 Circular Buffer

Circular Buffer is the way of carrying in put and out put by setting the size of Buffer and composing Buffer as a model.

Start point and End point are used to indicate Frame's site. Multithreading itself is possible to accomplish completing buffer and empting the buffer because it handle buffer by Start/End point.

And it brings improvement of capacity because Overhead is not happened.

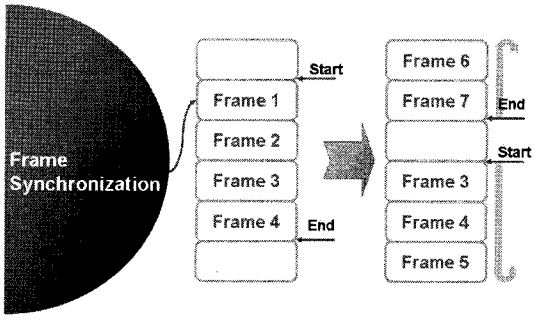


Figure 7 Circular Buffer structure

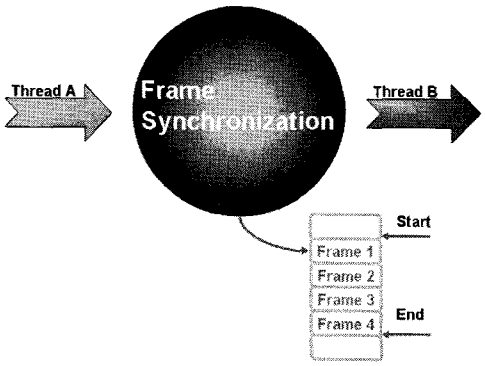


Figure 8 Thread working of Fame Synchronization

As the figure 8 is showing when the two Threads are acting Frame Synchronization is acting as showing the figure 9.

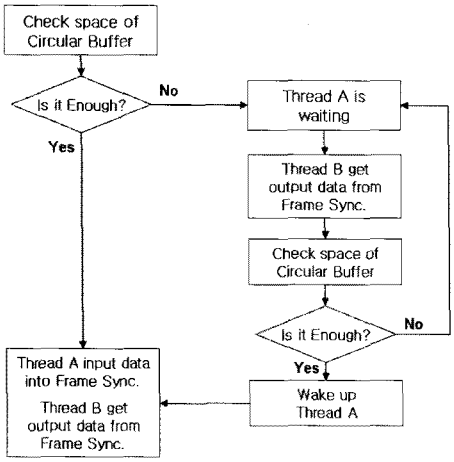


Figure 9 Block Diagram of Frame Synchronization

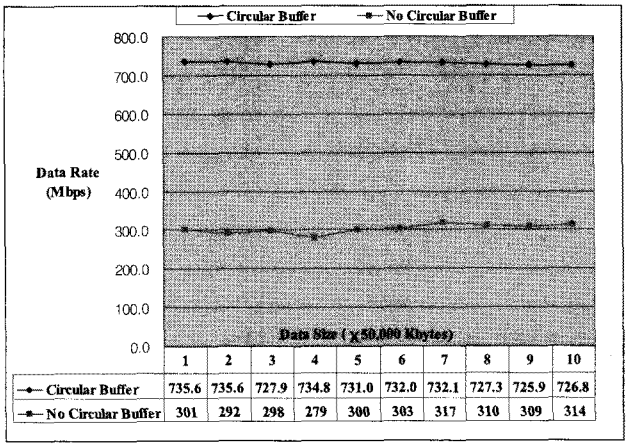
6. PERFORMANCE ASSESSMENT

The processing speed for (1) Frame Synchronization Software with SSE2 and (2) Frame Synchronization Software without SSE2 were measured. The computer specification for test was shown in Table 2.

Name	Description
CPU	Intel Core2 Duo E6700
Memory	2GB
OS	Windows XP

Table 2 Test Environment

256-byte of Frame data were used for input test data. Input file was made of multiple number of 256-byte frame data. Input file data was fed into software using configurable speed from 48.8Mbytes to 488.3Mbytes for simulating real operational environment.



	1	2	3	4	5	6	7	8	9	10
Circular Buffer	735.6	735.6	727.9	734.8	731.0	732.0	732.1	727.3	725.9	726.8
No Circular Buffer	301	292	298	279	300	303	317	310	309	314

Figure 7 **Compare Between Applied SSE2 and Not-applied SSE2**

From test results, Average speed for Circular Buffer case reached to about 730.9Mbytes while the speed for non-Circular Buffer case reached to 302.3Mbps.
This means Circular Buffer case shows 2.4 times higher performance than non-Circular Buffer case

7. CONCLUTION

This paper shows the optimizing method for frame synchronizing output data by the software using circular buffer and the solution of the cause of overhead problem by using the existing method. The high speed software frame synchronizer with SSE2 can provide several benefits like expandability and update which is, in fact, inherent to software. And Multithreading is possible to prevent Overhead caused by an assigning the memory and cancellation, by applying Circular Buffer in the output data that is Frame Synchronization.

This study can be applied for communication with satellite and in other digital communicating field where the frame Synchronization and Multithreading are needed.

8. REFERENCES

[1] Sang-Il Ahn, In-Hoi KOO, Tae-Hoon Kim, Young-Bo Sakong/ High-speed software Frame synchronizer using sse2 technology
[2] The Software Vectorization Handbook, Aart J.C.Bik
[3] IA-32 Intel® Architecture Software Developer's Manual Volume 2A: Instruction Set Reference, N-Z
[4] CCSDS, 'TM Synchronization and Channel Coding', Issue 1, Sep. 2003