# Design Study for the Development of Channel Coding Layer for a Packet Telecommand Decoder Based on CCSDS Recommendations for Satellite Communication on FPGA

**Haifa Yousuf Al-Bastaki**
Mohammed Bin Rashed Space Centre
Dubai, United Arab Emirates
Haifa.albastaki@mbrsc.ae

**ABSTRACT**

In the Bus System of any Communication Satellite, there exists a Telecommand (TC) Subsystem which plays a critical role in ensuring that telecommands sent from the ground are correctly received, decoded, and processed by the satellite. Following the recommendations of the Consultative Committee for Space Data Systems (CCSDS), a layered model is used to design a decoder system for any Packet Telecommand Decoder (PTD), which is central to the Telecommand data process. The benefit of adhering to CCSDS standards is the ability to operate the satellite from any standardized CCSDS ground station, enhancing cross-support capabilities. This study aims to explore the development of the Channel Coding Layer according to CCSDS recommendations, with the focus on designing the programming logic for the decoder system, which manages services related to synchronization and error correction and detection in transmitted data. The paper details the algorithm design logic of two core elements within the layer: 1) the Finite State Machine (FSM) controller which provides synchronization services to control the layer's decoding process 2) the Decoder, designed using Modified BCH (63,56) (Bose-Chaudhuri-Hocquenghem) code, which operates using single error correction and double error detection mode (SEC). The designed flowcharts and block diagrams illustrate the data flow and operational steps of the overall decoding process. This systematic approach to designing and organizing the decoding algorithm logic lays the groundwork for programming the channel coding layer using VHSIC Hardware Description Language (VHDL) and testing it on Field-Programmable Gate Array (FPGA) in future work. This paper uniquely addresses the gap between the CCSDS recommendations and the practical implementation by translating textual guidelines and recommendations into a structured, executable, and actionable design approach! It also implements and tests the state control logic algorithm modeled for the channel coding layer using VHDL.

***Keyword: -*** *Packet Telecommand Decoder, Channel Coding Layer, CCSDS, Finite state machine, BCH (63,56) code, Decoder.*

## 1 INTRODUCTION

The Channel Coding Layer is one of the essential layers to consider when designing a Packet Telecommand Decoder (PTD) for communication satellites. According to [4], several CCSDS layers were implemented to develop a PTD capable of fully performing the receiving-end functionality of a telecommand decoder system. The system architecture of this single-chip implementation of the TC Decoder Core represents the integration of several CCSDS layers. The internal architecture of the PTD described in [4] addresses five layers dedicated to the Synchronization and Channel Coding Sublayer and the Data Link Protocol Sublayer of the CCSDS protocol, as shown in Figure 1. One of the layers is the Channel Coding Layer which is the first layer encountered once the telecommand has been successfully received and bit modulation is detected in the physical layer. This study focuses on the detailed design of the Channel Coding Layer core, whose primary objective is to decode and process only error-free protocol data units (PDUs) for the above layers. The design of this layer relies on two main components: the Finite State Machine (FSM) controller and the BCH (63,56) decoder, both essential to its functionality. The following sections model the programming logic design of these core elements according to CCSDS standards, outlining a plan of action to bridge the gap between theoretical guidelines and real-world implementation. By transforming recommendations into visual roadmaps, this approach aims to promote logical thinking, break down design challenges, and apply structured solutions to the design of the Channel Coding Layer.

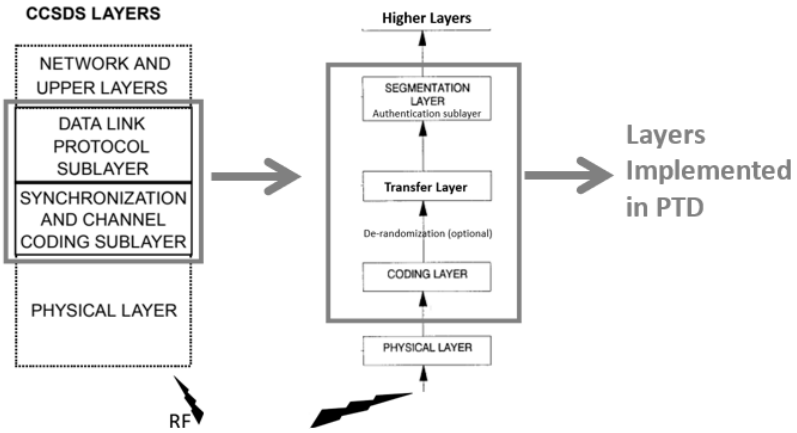



*Figure1. CCSDS Layers Implemented in the PTD*

### 1.1 Channel Coding Layer Overview

The channel coding layer plays a crucial role in ensuring that the symbol streams of the received telecommands from the channels are correctly structured according to CCSDS protocols. It also applies error correction and detection techniques to guarantee that only error free data is passed to the next layer. As outlined in [3], the overall "Reception Logic" of the channel coding Layer is represented as a Sate diagram with 3 main states which are 1) INACTIVE STATE 2) SEARCH STATE and 3) DECODE STATE. Thus, the design of a finite state machine is necessary to fulfill the intended functionality of this layer. While the principal objective of the channel coding layer is to decode the channel symbol streams, which happens in the DECODE STATE (the final state), it is also responsible for performing synchronization services such as checking if the channel is active, Coded symbol stream is available, and the start of the Protocol Data Unit (PDU) is recognized. It is important to note that the decoding of the telecommand symbol streams cannot begin unless the layer is in the DECODE STATE. At this point, the BCH (63,56) decoder is used to decode the channel's coded symbol streams in Single Error Correction (SEC) mode. This mode allows the decoder to correct one-bit errors and detect two-bit errors.

### 1.2 Channel Coding Layer Protocol Data Unit (PDU)

As shown in figure 2, the physical Layer is limited to detecting Acquisition and Idle Sequence, and separating Command Link Transmission Unit (CLTU) from the TC packets at the receiving end. It then passes those CLTU's to the channel coding layer. Therefore, The CLTU is the Protocol data unit (PDU) of the channel coding layer.

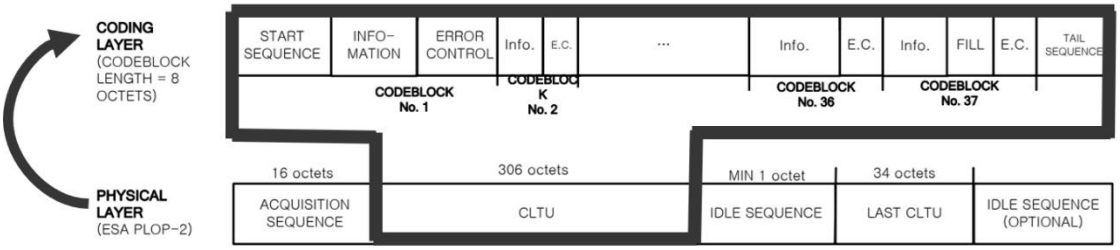



*Figure2. Protocol Data Unit of Channel Coding Layer [4]*

Each CLTU (Command Link Transmission Unit) represents a frame and consists of a start sequence, multiple code blocks, and a tail sequence, as shown in the figure below. It is important to be familiar with the PDU of the layer to ensure correct data processing, compatibility, and adherence to CCSDS protocol standards in the design.

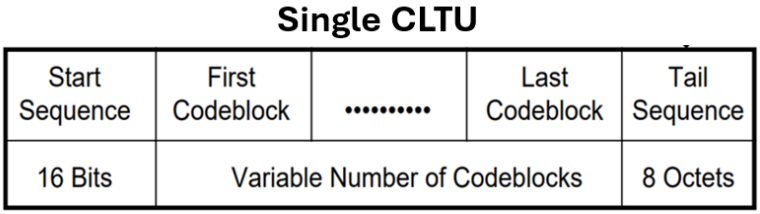



*Figure3. Command Link Transmission Unit structural components [4].*

At the receiving end, the telecommand codewords will be decoded using the BCH (63,56) decoder for this paper's design. Therefore, CCSDS recommends that the CLTU have the structural components shown in Figure 3 for the chosen decoder [3]. The CLTU must consist of the followings: -

- **Start Sequence**: A unique, fixed-length pattern of 16 bits (2 octets). The pattern is 0xEB90 in hexadecimal, indicating the start of a CLTU.
- **Tail Sequence**: A unique, fixed-length pattern of 64 bits (8 octets). The pattern chosen is (555555555…) in hexadecimal representing alternating zeros and ones in binary as described in [4]. The pattern must end with a binary "1" to form a "5" in hexadecimal. This unique pattern is constructed to force the decoder to recognize the end of the CLTU.
- **Code block**: Code blocks are placed between the Start and Tail sequences and contain both information and error control field bits, as shown in the figure below. A single code block bit length is fixed to 64 bits as it is the most efficient of the options and the recommended standard according to CCSDS [1]. The maximum number of code blocks that can be sent through a single CLTU may vary, as described in [3] & [1]. For this study, the code block bit length is set to the standard recommended bit length n=64 bits, and the maximum number of code blocks that can be sent through a CLTU is chosen to be 37 as in [4].

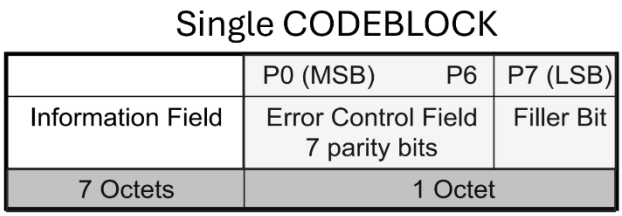



*Figure4. Code block components [4]*

Each code block (8-octet ➔64 bits) allocates 1 octet to the error control field and the remaining 7 octets to the information field. These bits are generated at the sending end using a systematic block coding procedure, which processes 56 information bits and generates 7 parity check bits used for error control at the receiving end (per code block) [3]. The last bit (LSB) in the error control field, known as the filler bit, is appended to complete the 8-bit error control field and serves to distinguish between information and tail code blocks. This is because a tail sequence block will always have a filler bit = '1' whereas an information code block will always have a filler bit = '0'.

### 1.3 Channel Coding Layer Block Diagram – Design Description

The channel coding layer block diagram in figure 5 describes the inputs and outputs of our designed layer block and provides a high-level understanding of how the overall core will operate. The inputs to the design block are system clock signal, reset signal, and several channel lines. Each channel will consist of 3 input lines which are:

- Symbol Stream signal (TCS)
- Active Signal (TCA)
- Channel Clock Signal (TCC)

Similarly, these channel input signals have been derived from the design in [4] for the channel coding layer. Table 1. provides the signal description of each channel input line.

Table 1. Channel Input Signal Lines to the Channel coding Layer block Design

| Signal Name | Signal Description |
|---|---|
| Channel Active Signal (TCA) | Represented as either “1’ or “0” and serves as an enable signal to indicate that the channel is active for the availability of the symbol stream. |
| Channel Clock Signal (TCC) | Represents the channel clock signal. This signal can only be recognized while TCA is asserted high |
| Channel Symbol Stream signal (TCS) | Represents the data sent through the channel. The bit data is valid at every falling edge of the channel clock signal. |

This paper’s design will be limited to 3 channel inputs. Therefore, total of 6 input lines from the channels will be inputted in total. However, the channel coding layer implemented in [4] can receive up to 6 channels. Moreover, The Outputs of the design block are described in Table 2 at the end of section 1.4. The designed coding layer will generate signals to provide feedback on the core’s performance and to monitor if the design is functioning as intended.

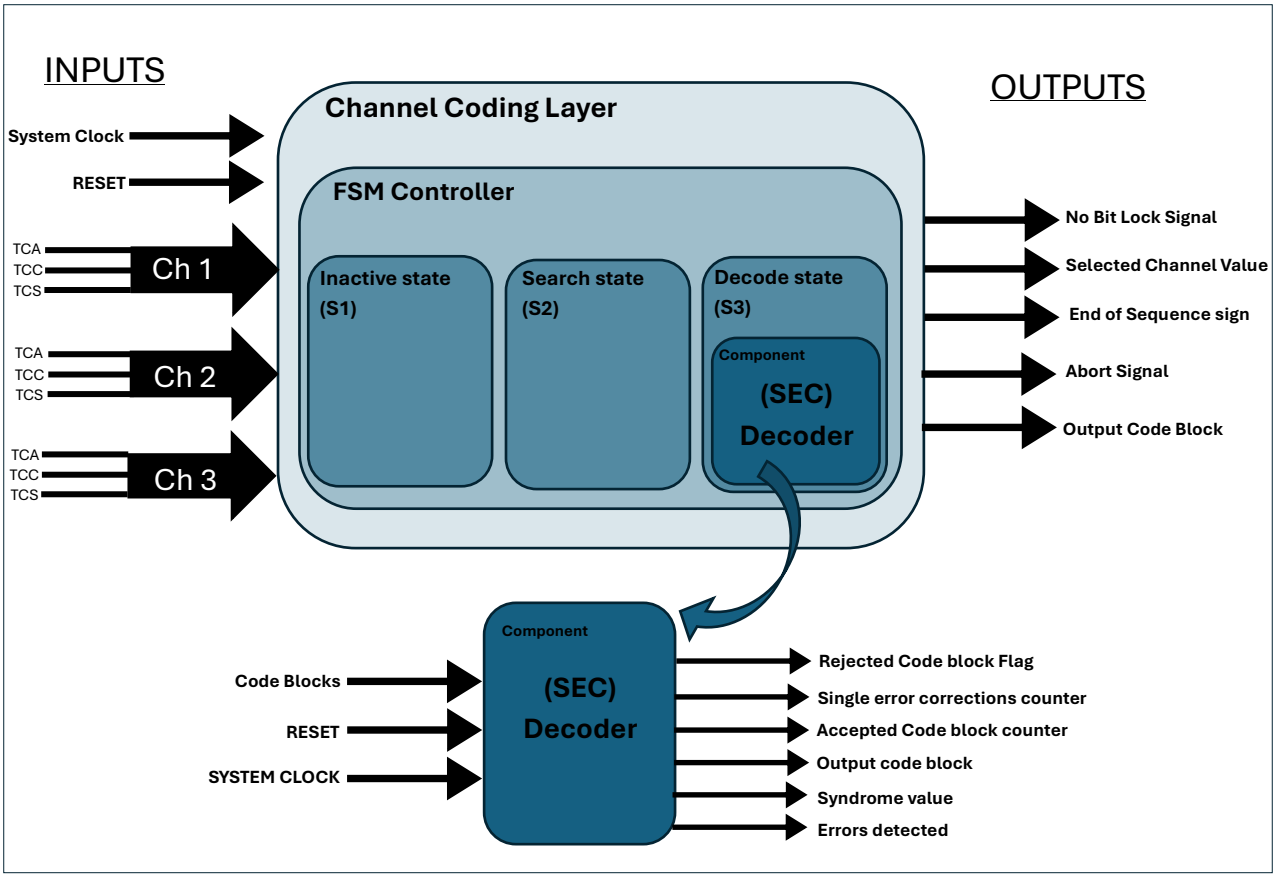



*Figure5. Channel Coding Layer Block Diagram*

The design study of the channel coding layer decoding process will be divided into two main parts: 1) the Finite State Machine (FSM) and 2) the SEC Decoder. The SEC Decoder will be treated as a component that operates when the FSM is in the Decode state. The first objective is to design the programming logic for the FSM to ensure that the channel is active and the CLTUs are recognized, allowing the proper retrieval of code blocks following CCSDS recommendations (synchronization services). Once FSM transitions to final state (DECODE) and the coded symbol streams are correctly sampled, the SEC decoder can then begin the decoding process, with the code blocks passed to it. As such, the SEC Decoder will be designed as a separate component. Finally, to achieve the full functionality of the channel coding layer, the two designs will be integrated and used together in future work.

### 1.4 Finite State Machine Programming Logic Design Methodology & Modelling

The Finite State Machine (FSM) design follows the CCSDS-recommended procedures for receiving a CLTU at the receiving end [1]. The FSM will consist of a finite number of states and transitions triggered by specific inputs, referred to as "events." The FSM states are: 1) Inactive-S1, 2) Search-S2, and 3) Decode-S3. The operation of the channel coding layer's FSM has been outlined in high-level textual guidelines and recommendations by the CCSDS. Now, it's time to translate those descriptions and arrange them into actionable and executable design to develop a state control logic for the Layer's decoding process!

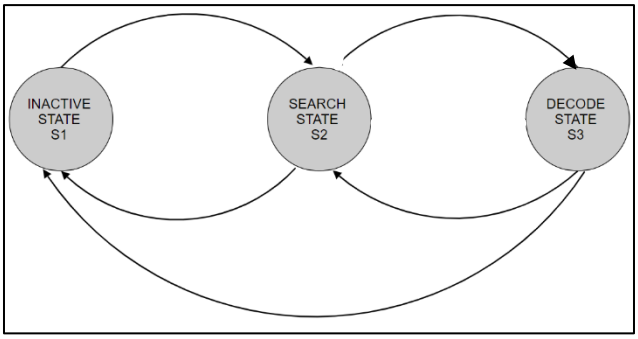



*Figure6. Channel Coding Layer Sate Diagram*

#### 1.4.1 INACTIVE State (S1): Operation principle and Logic Flowchart

This is the idle/default state of the FSM. In this state, the process of detecting an active channel signal (TCA) occurs simultaneously across all three channels, meaning it monitors TCA1, TCA2, and TCA3 at the same time. As outlined in Table 1, the TCA signal acts as an enable signal that indicates the channel is active and that symbol streams are available on that channel. Therefore, when $TCA_i$ = "1" for any channel, it means that channel is **activated**; if TCAi = "0," it indicates that the channel is **deactivated**. Initially, the TCA signal is active low. It is important to note that all channels will have equal priority in this state for the design of this paper.

To clarify, the FSM will monitor three separate channels simultaneously. Once a TCA signal is detected as "high" on one or more channels, the FSM will transition to the next state, the **Search State**. Multiple channels can be activated at the same or different times. Regardless of when the channels become active, even if not all channels activate at the same instant, the FSM will still transition to the next state for the respective active channel.

Upon detecting an active channel, the FSM will output a signal called "NO_bit_lock" and set it to "0". Initially, this signal is active high, indicating that no channel is active, but it goes low once any $TCA_i$ signal is detected. Below flowchart summarizes the process of the INACTIVE State.

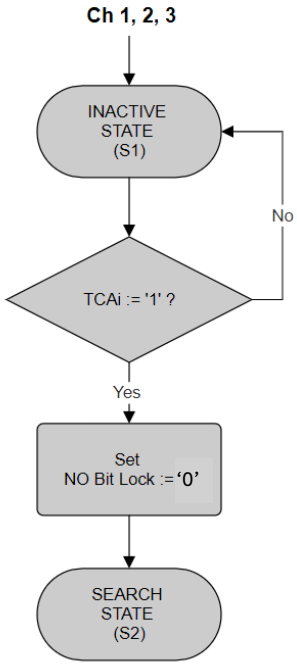



*Figure7. INACTIVE State (S1): Logic Flowchart*

### 1.4.2 SEARCH State (S2): Operation principle and Logic Flowchart

When the FSM Transitions to SEARCH state for any channel, The "TCA" of the Activated channel shall always remain high TCAi='1', else the FSM transitions back to the INACTIVE STATE for that channel. The No Bit Lock signal is set to '1' when all channels are deactivated therefore if at least one channel is active and the remaining got deactivated in S2 the signal stays '0'.

The FSM in this state will start searching for the Start Sequence pattern "0xEB90" SIMULTANOUSLY on all activated Channels. The channel Symbol Stream signal (TCS) is detected on every falling edge of the channel clock signal (TCC) as described in [4].

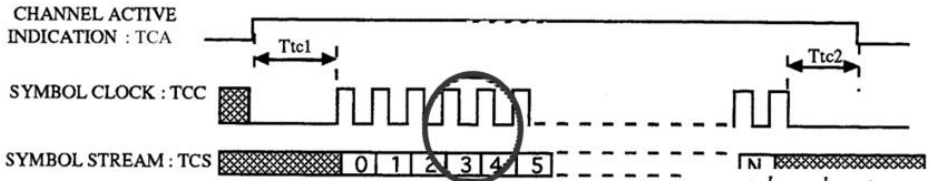



*Figure8. TCS bits Sampling process [4]*

Paper [1] provides two options in the CLTU reception procedure for detecting the start sequence in relation to error handling. Option 1 requires strict detection with no errors allowed in the start sequence. Option 2 allows for a single error in the start sequence while still considering it valid. Since SEC mode was selected for the channel coding layer design, CCSDS recommends using Option 2. Allowing a single error in the start sequence increases reliability by reducing the probability of rejecting valid start sequences, especially in designs with Bit Error Rates (BER) of $10^{-5}$ or higher. Therefore, we will proceed with option 2 in which the designed FSM will allow only one bit error anywhere within the 16-bit start sequence pattern 0xEB90: = '1110 1011 1001 0000' and still consider it correct!

In [4], the coding layer core design also allows the detection of the inverse of the start sequence, which is 0x146F in hexadecimal, while permitting a single bit error in the inverted sequence. According to the [1], the reason for enabling detection of both the start sequence (0xEB90) and its inverse (0x146F) is to address data ambiguity in the received bit stream (with respect to ‘0’ and ‘1’) that may be introduced due to non-return-to-zero (NRZ) encoding in the incoming start bit streams. The physical layer can typically resolve this ambiguity based on the modulation scheme used. However, if this ambiguity is not resolved at the physical layer, CCSDS recommends handling it by searching for both the start sequence and its inverse, allowing a single bit error in each, and still considering them valid. For the design in this paper, it is assumed that the data ambiguity is resolved by the physical layer, and the designed core will only search for the start sequence "0xEB90” allowing one bit error anywhere.

Moreover, In the Search State (S2), only one channel is selected for the Decode State (S3), while the others are discarded. This means that the search occurs simultaneously across all active channels until a start sequence is detected on one channel. Once the start sequence is found on a certain channel, that channel is the only one to move to the Decode state, and the remaining channels are discarded until the next channel selection occurs. The FSM will lock the selection to the first channel where the start sequence is detected and discard the others. After detecting the start sequence and locking onto one channel, the FSM transitions to the next state, the Decode State (S3).

Upon transitioning to S3, the FSM outputs two signals:

1. DECODE_SIG: This signal goes high following the detection of the start sequence, indicating that the state transition will occur. (This will be an internal signal in our design)
2. SELECTED_CHANNEL_VALUE_SIG: This is a 3-bit vector that represents the value of the selected channel where the start sequence was first detected.

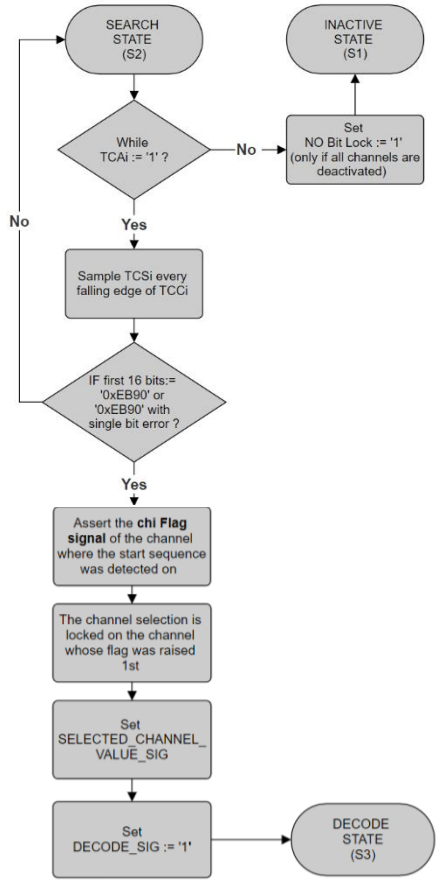



*Figure9. SEARCH State (S2): Logic Flowchart*

### 1.4.3 DECODE State (S3): Operation principle and Logic Flowchart

The Decode State (S3) represents the final state of the Channel Coding Layer FSM. In this state, the bit streams are decoded using the SEC decoder. However, aside from the actual decoding process, the state must also manage several other tasks to prepare the data for decoding and ensure that the process follows CCSDS recommendations. This section will focus on the control logic for this state within the FSM, treating the SEC decoder as a black-box component. As mentioned earlier, the SEC decoder will be designed separately and later linked to the FSM design.

During this State, the FSM will receive the continuous bit stream and treat every 64 bits as a single code block, which aligns with the format structure shown in Figure 4. Therefore, each 64-bit segment of the bit stream will be extracted and organized into a distinct code block before being passed to the SEC decoder for further processing.

While the FSM is in the Decode State, it will continuously monitor several conditions. If any of these conditions are triggered, the FSM will be forced to exit the Decode State and halt the decoding process. These conditions are critical for ensuring that the system operates correctly and reliably during decoding.

1. The **TCA** signal of the selected active channel must always remain high ('1'). If, at any time during the Decode State (S3), the TCA signal goes low ('0')—indicating channel deactivation—the FSM will transition to the **Inactive State** and send a signal to abort the entire octets, assuming any code blocks had already been sent to the upper layer. At this point, the NO_bit_lock signal is set to '1', and the entire cycle is repeated for all channels.

2. If a timeout occurs on the channel clock signal (TCC) of the selected active channel (i.e., TCC is not detected for a certain period), the FSM will transition back to the **Search State** and send a signal to abort the entire octets, assuming any code blocks had already been sent to the upper layer. The timeout value can be determined based on your design parameters, such as system clock frequency, channel clock period, baud rate and other factors. This timeout mechanism is based on the method described in [4] and serves as a protection measure to reactivate the channel selection mechanism, ensuring the FSM does not remain locked on a channel without a clock signal in the event of a receiver breakdown.

3. If the **Tail SEQUENCE** is encountered, which has the same size as a code block (64 bits) but is specifically constructed to force the decoder to stop, the FSM will transition back to the **Search State** and send a signal indicating the end of the code block transfer. The **Tail SEQUENCE** is a unique pattern explained in section 1.2. In this case, there is no need to send an abortion signal as this simply indicates the normal end of the CLTU transmission.

4. If more than 37 code blocks were received and no **TAIL Sequence** has been encountered, the FSM transitions back to the **Search State** and sends a signal to abort the entire octets, assuming any code blocks had already been sent to the upper layer. This occurs because the size of the code blocks within the CLTU has been predefined for this design (section 1.2). Therefore, based on the size you have set for the code blocks in the CLTU, this mechanism shall apply accordingly.

5. If the “single error corrected” code block outputted from the SEC decoder had the last bit, the filler bit, as '1', it will be considered as a Tail sequence. The FSM transitions back to the **Search State** and sends a signal indicating the end of the code block transfer. In this case, there is no need to send an abortion signal as this simply indicates the normal end of the CLTU transmission. (note: this case doesn’t apply on error-free code blocks outputted from the SEC decoder)

6. If the code block rejection flag from the SEC decoder is ‘0’ which indicates that the code block passed to it is found uncorrectable (more than single bit error was found), the FSM transitions back to the **Search State** and sends a signal indicating the end of the code block transfer. In this case, there is no need to send an abortion signal as this simply indicates the normal end of the CLTU transmission.

For case number 5, it should be noted that the SEC decoder ignores the filler bit during decoding (further discussed in section 1.6). Therefore, this technique (Case 5) is used to significantly reduce the likelihood of missing a tail sequence, but it comes with the drawback of a slightly increased frame rejection rate. This occurs because an information code block may be error-free after correction, but a bit flip in the filler bit could cause the block to be rejected (recall that all information code blocks filler bits are set to ‘0’ during transmission). Under this algorithm, such a code block would be rejected even though the information is correct. Without this algorithm, the erroneous filler bit would not be checked,

and the code block would be accepted. Although this method is optional, it has been included to avoid missing a tail sequence, which could otherwise lead to missing the next CLTU [1] [5].

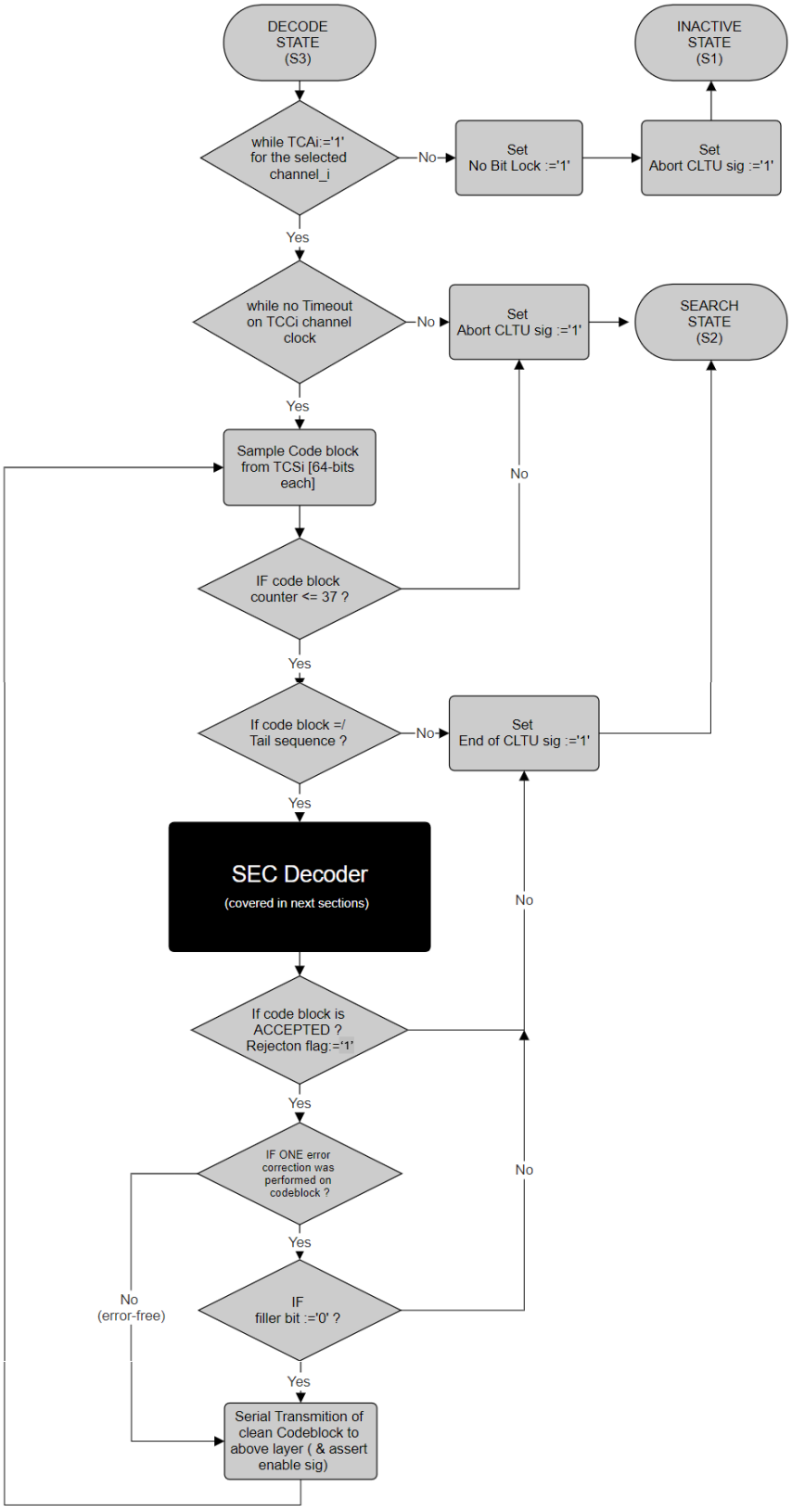



*Figure10. DECODE State (S3): Logic Flowchart*

### 1.4.4 Coding Layer Block Top Entity Input and Output ports

Referring to the outputs mentioned in the **Coding Layer block diagram** in Figure 5, these signals have been summarized in the below table. Gaining a clear understanding of the state control logic for each state of the layer in section 1.4 will help relate and interpret the output signals summarized in Table 2, as well as understand their expected behavior during the operation of the code.

Table 2. Output Signals from the Channel Coding Layer Block

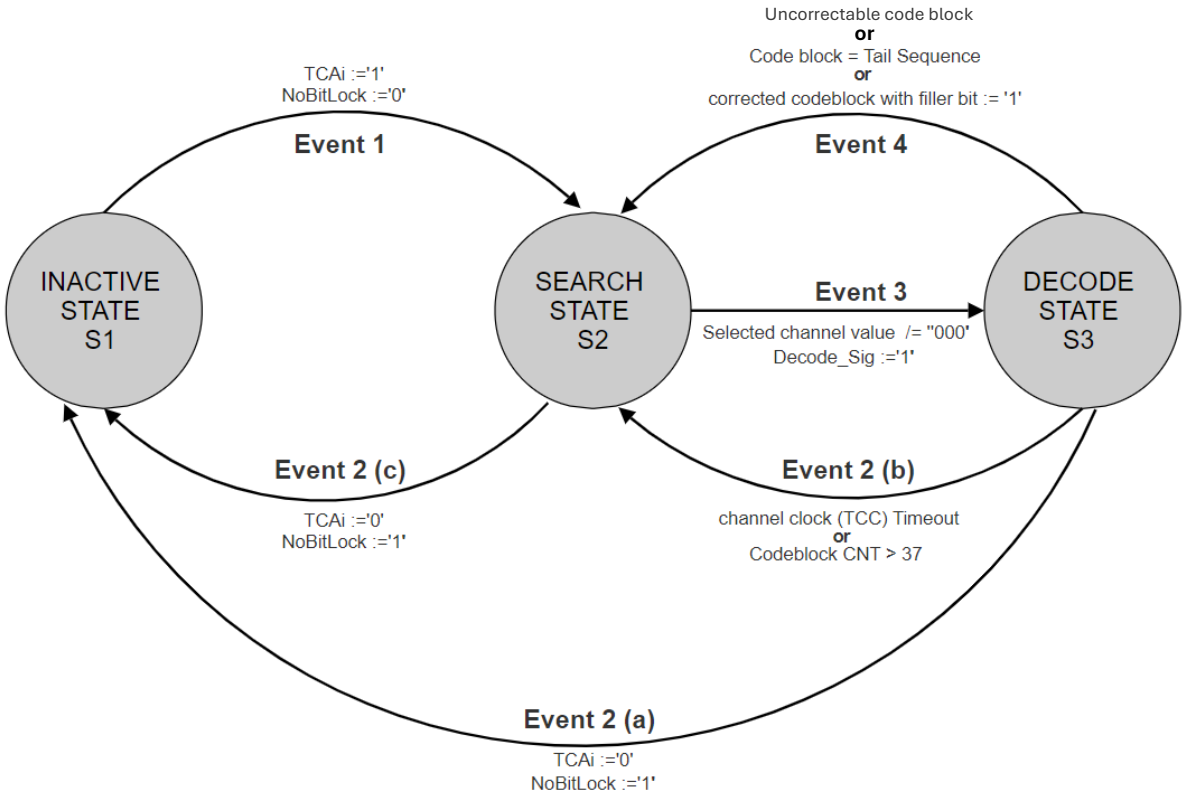
| Signal Name | Signal Description |
|---|---|
| No Bit Lock Signal | Represented as either “1’ or “0” and serves as an enable signal to indicate that a channel activation has occurred. This signal goes Low following a detection of a TCAi:=’1’ and goes high following a channels deactivation TCAi:=’0’. |
| Selected Channel Value signal | Represents the value of channel that has been selected. |
| End of Sequence Sign Signal | This signal is asserted to indicate normal end of the CLTU transmission. |
| Abort Signal | This signal is asserted to signal the upper layer to abort the entire octets, assuming any code blocks had already been sent. |
| Output Code block | Represents the outputted code blocks |

```
library ieee;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use work.code_blocks_package.all;

entity Coding_Layer_data_stream_sampler  is
    port (
        CLK_MHZ                 : in  std_logic; -- System Clock
        RSTn                    : in  std_logic; -- Reset on 0

        TCA                     : in  std_logic_vector(2 downto 0);
        TCC                     : in  std_logic_vector(2 downto 0);
        TCS                     : in  std_logic_vector(2 downto 0);

        NO_BIT_LOCK             : out std_logic;
        SELECTED_CHANNEL_VALUE  : out std_logic_vector(2 downto 0);
        END_OF_SEQUENCE_SIGN    : out std_logic;
        ABORT                   : out std_logic;
        CODE_BLOCKS_O           : out code_block
    );
end Coding_Layer_data_stream_sampler;
```

*Figure11. Coding Layer implemented Design Entity I/O Ports- VHDL Script*

### 1.4.5 Channel Coding Layer State Machine Diagram

The below showcases the final state diagram, illustrating the critical event instructions that trigger transitions between the three states.




| Event | Event Description | Transition |
|---|---|---|
| Event 1 | CHANNEL ACTIVATION | S1 ➔ S2 |
| Event 2 (c) | CHANNEL DE-ACTIVATION | S2 ➔ S1 |
| Event 3 | Selection of the first channel with the Start Sequence | S2 ➔ S3 |
| Event 4 | Tail Sequence Detection<br>Filler bit :='1' for corrected code block<br>Uncorrectable code block | S3 ➔ S2 |
| Event 2 (b) | > 37 code blocks were accepted<br>Timeout on the TCC- channel Clock signal | S3 ➔ S2 |
| Event 2 (a) | CHANNEL DE-ACTIVATION | S3 ➔ S1 |

*Figure 12. Channel Coding State diagram Description*

## 1.5 Implementation, Results, and Discussions - (Control State Logic Design)

The proposed model of the state control logic of FSM for the coding layer (for 3 channels) has been implemented using VHSIC Hardware Description Language (VHDL) and simulated in ModelSim. A test bench was also generated to verify the functionality of the design. All events depicted in Figure 12 have been tested to ensure that the FSM transitions correctly between the mentioned states. Note that the SEC decoder has been treated as a black-box component and has not yet been implemented in this design. The primary objective of this simulation was to test the receiving end for input synchronization, channels activation, start sequences detection, data stream sampling, and channel selection, along with all critical events previously mentioned that cause state transitions. Design parameters such as system clock frequency, channel clock frequency, and channel baud rate have been selected for this simulation based on the specific requirements of the intended design.

### 1.5.1 Waveform Simulation Results using MODELSIM

#### 1.5.1.1 Tests Performed on One Channel (Ch 0) for Proof of Concept.

- **Results for Tests E1 and E3 and E4 (S1➔S2➔S3➔S1)**

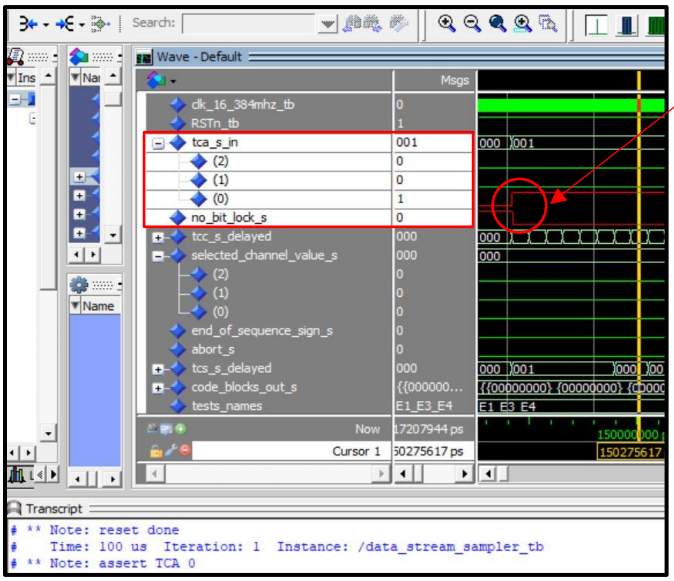



*Figure 13. Event 1- Channel Activation on channel 0 Waveform*

Shown Above are the waveform results when Event 1 is triggered which is when channel 0 becomes activated. As TCA(0):=’1’ we expect the NO Bit Lock signal to go low and this transitions the system to Search State where the search for start sequence will take place.

Expected results: TCA(0):=’1’ ➔ NO_BIT_LOCK = 0 ➔ S1->S2.

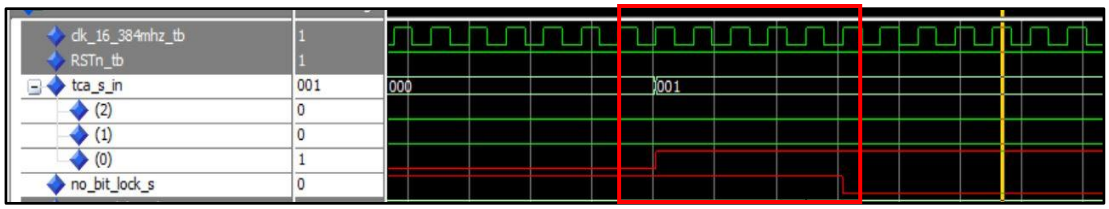



*Figure 14. Event 1- Channel Activation on channel 0 Waveform*

Note that in this design, As TCAi signal goes high, sufficient time is given before No Bit Lock signal goes to “0” to avoid issues caused by signal noise or glitches. This prevents us from mistakenly detecting a TCA signal that is not actually asserted and ensures its valid, not just a result of noise.

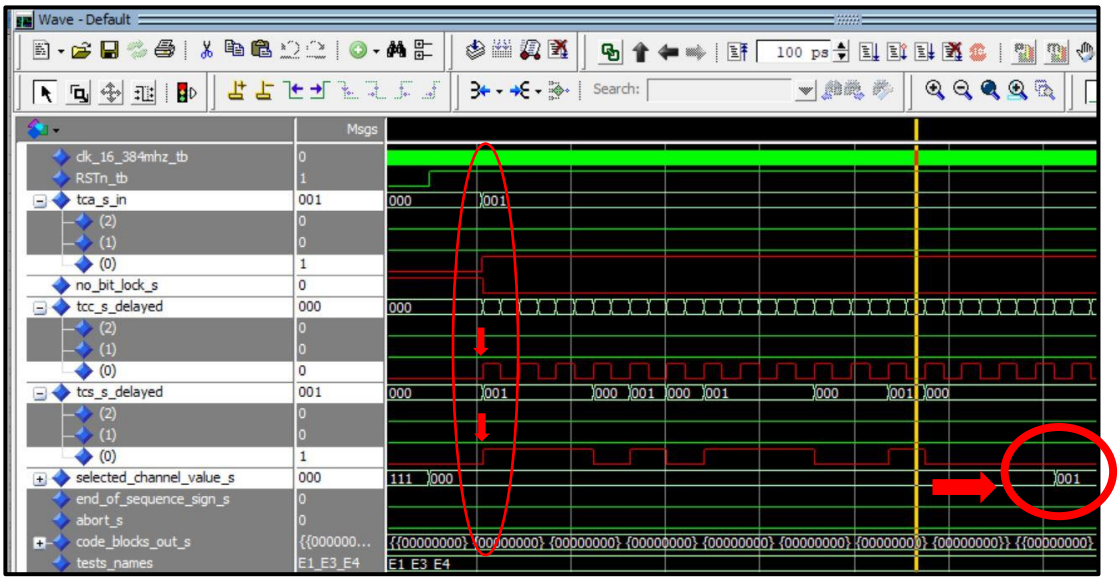



*Figure 15. Event 3 – Start Sequence Detection on channel 0*

Following the detection of channel 0 TCA in figure 13, the FSM transitions to Search state (S2). **Search State (S2)** is illustrated clearly in Figure 15, where the channel clock signal (TCC) and the channel symbol stream signal (TCS) appear as soon as the No Bit Lock signal goes low, indicating that we have entered S2. In this state, the system searches for the start sequence by sampling the TCS signal on every falling edge of the TCC and the rising edge of the system clock. In this scenario, the first 16 incoming bits received were exactly “0xEB90”. Once the start sequence is detected on channel 0, the **Selected_Channel_Value_Signal** is set to “001”, indicating that channel 0 has been selected. This event triggers the FSM to transition to the **Decode State (S3)**, locking the selection onto channel 0.

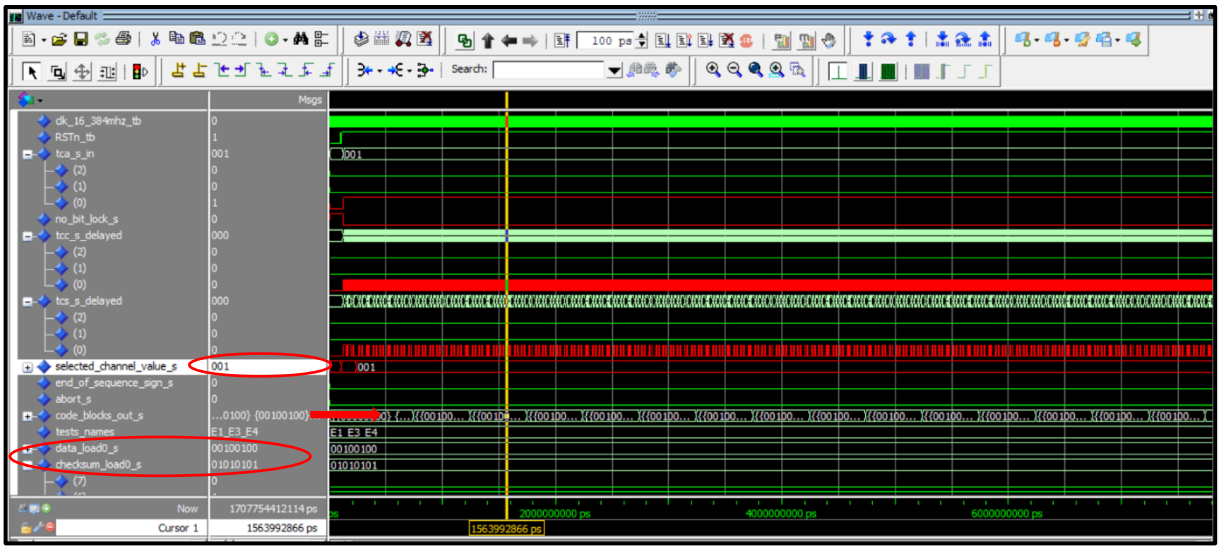



*Figure 16. Channel 0 Transition to Decode State (S3)*

The figure above shows the waveforms in the **Decode State**. The **selected_channel_value_S** signal indicates channel 0 is selected. In this scenario, **Data_load_s** represents the transmitted bits, which is "0x24" repeatedly, and similarly, **checksum_load_s** represents the tail sequence pattern. The incoming data will be sampled from the **TCS** signal and packed into octets, with every 8 octets forming a code block. These code blocks are continuously sampled and transmitted through the **code_block_out_Signal** until an event triggers a state change. Note that the **SEC decoder** has not been implemented yet, so the sampled data is directly outputted through the output signal assuming it successfully passed the decoder.

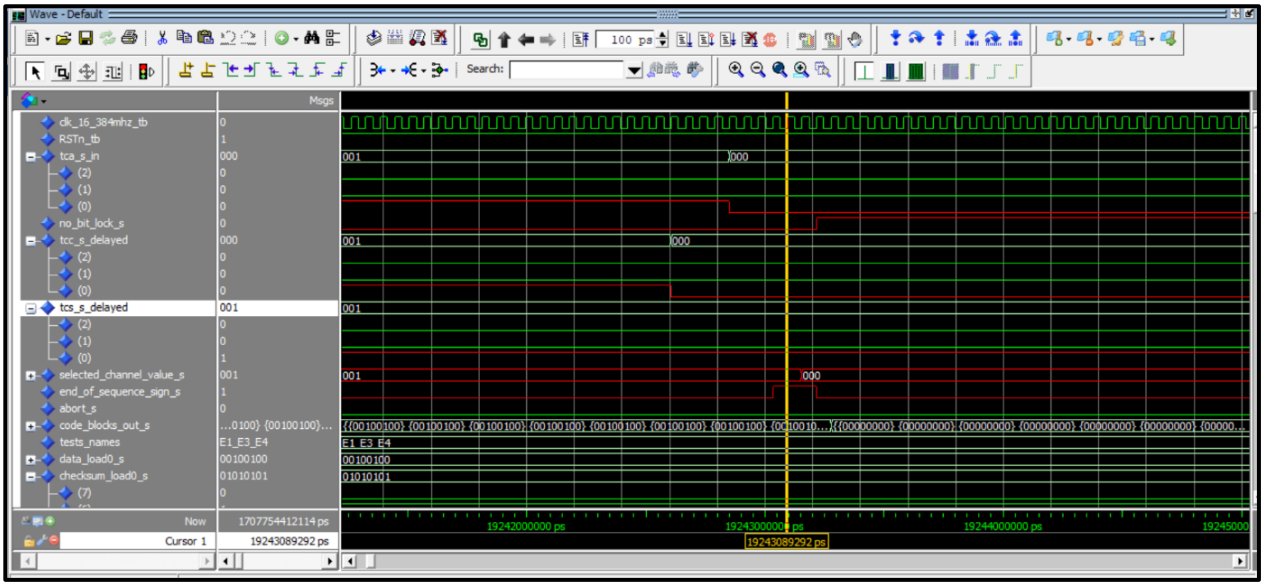



*Figure 17. Depicting Event 4 (Tail sequence) - Channel 0 Transition to S2 then to S1*

Figure 17 shows the waveforms for a section of a code added in our design. In this paper's design code, a section was added to check if any channel has been deactivated because it reached the end of overall transmission i.e end of CLTU (Command Link Transmission Unit) and the tail sequence has been detected. If the TCA signal for any selected channel goes low during S3, the system checks if the end-of-sequence signal (end_of_sequence_sign_s) is asserted which indicates the end of CLTU transmission. If it is, the system transitions to the search state while setting selected_channel_value_signal to "000" as shown above. Once it's transitioned to S2, because No TCA is detected search cannot take place and hence it'll transition back to the inactive state with Setting NO Bit Lock signal to "1".

- **Results for Test E2 (c)**

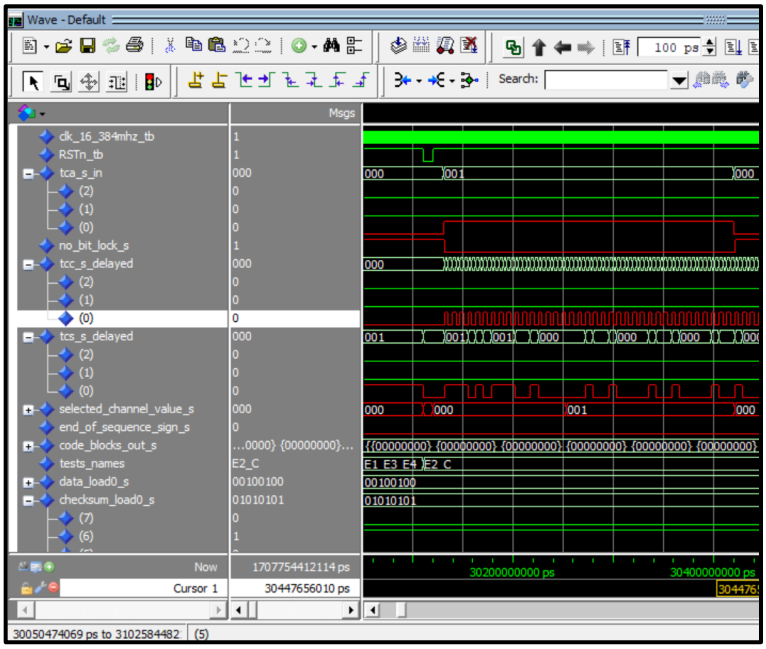



*Figure 18. E2(c) - Channel 0 Transition from S2 to S1 due to channel deactivation*

In test E2(c), TCA will go low in the middle of the search state. Expected result is for the system to transition from S2-->S1, therefore NO_BIT_LOCK: = 1 & Selected Channel Value:=000.

- **Results for Test E4 (Case 2: Assuming a correct code block with filler bit '1')**

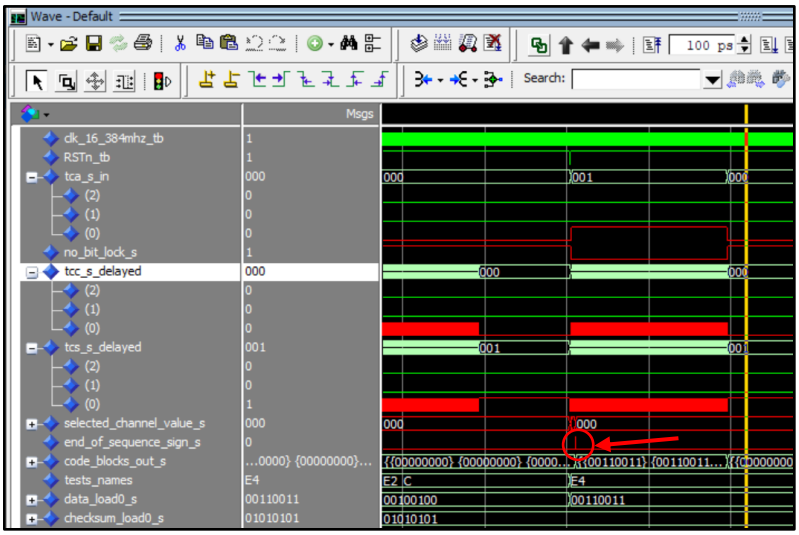



*Figure 19. E4 – Code block rejection due to filler bit: ='1'*

In this **E4 Test**, the data loaded is "0x33", which ends with a binary 1. Once the bits are sampled and packed into a code block, the last bit of the code block will be 1. As a result, **E4** is triggered, and the expected outcome is setting **END_OF_SEQUENCE_SIGN := '1'.**

- **Results for Test E2(b) – (Case 1: More than 37 code blocks, Case 2: Timeout on TCC )**

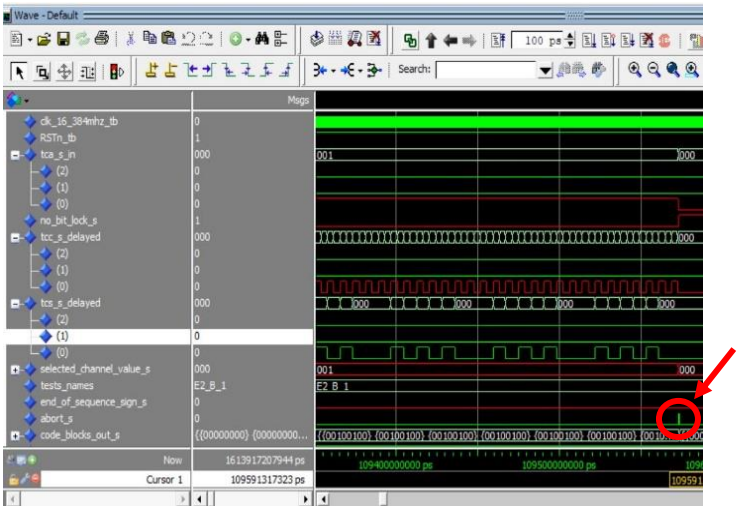



*Figure 20. E2(b) – Case 1 >37 code blocks were sent without encountering the tail sequence*

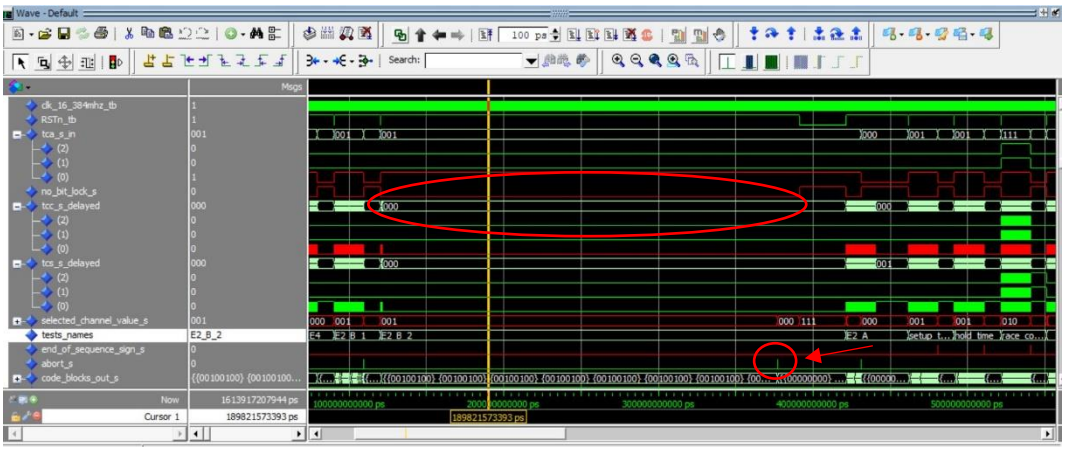



*Figure 21. E2(b) – Case 2: Timeout value on TCC channel 0*

In **Test E2(b) Case 1**, more than 37 code blocks were sent, indicating a checksum error as the last packet is not the end of the sequence. In **Case 2**, the **TCC** signal goes low and exceeds the timeout value set for this design. The expected outcome for both cases is for the system to transition from **S3 to S2**, with **ABORT** set to '1'.

- **Results for Test E2(a)**

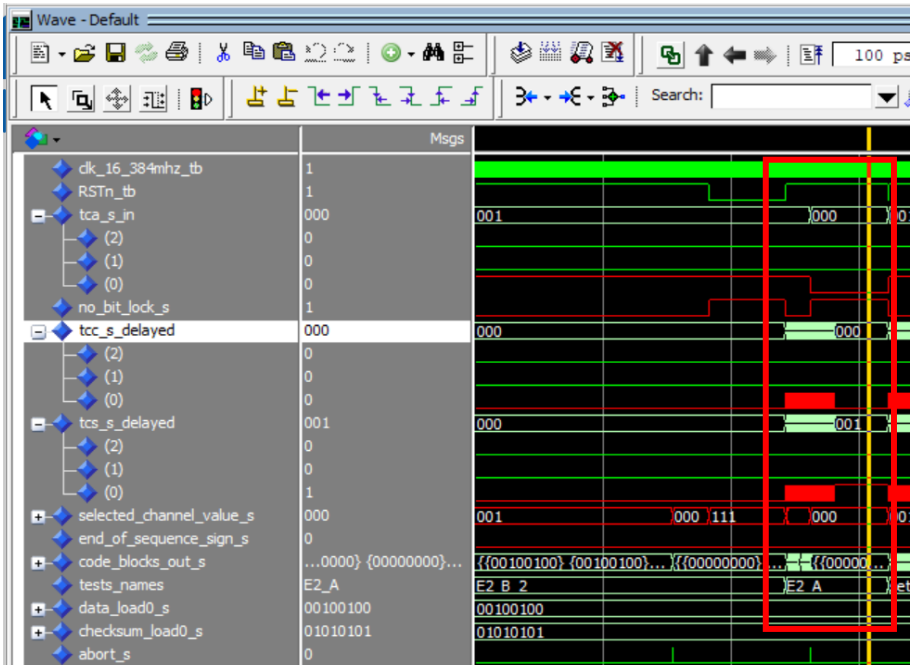



*Figure 22. E(a) – channel deactivation during Decode state*

In **Test E2(a)**, the **TCA** signal goes low during the middle of the "Decode State." The expected result is for the system to transition from **S3 to S1**, with **NO_BIT_LOCK** set to '1'.

### 1.5.1.2 Test for Race Condition on All Three Channels

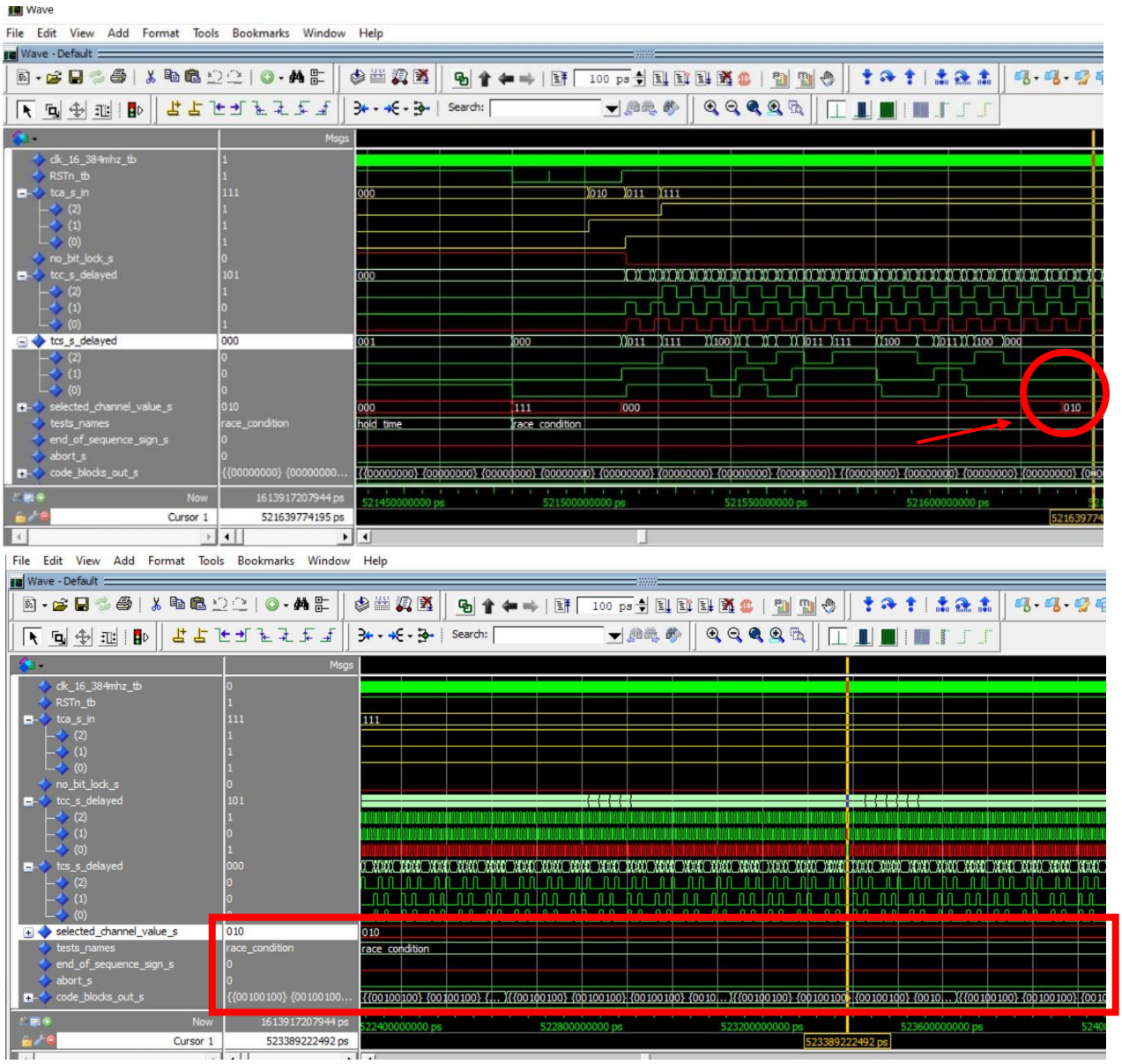



*Figure 23. Race Condition Test on All Activated Channels*

Figure 23 shows the race condition test performed across all channels. As clearly shown, Channel 1 wins the selection, with selected_channel_value_S set to '010'. This occurred because Channel 1 was the first channel detected with the start sequence. This can also be predicted from the waveforms, as the TCS signal first arrives on Channel 1, assuming the start bits are correct. As a result, the selection gets locked onto Channel 1, and as shown in the lower part of Figure 23, the outputted code blocks are sampled from Channel 1's data.

## 1.6 Single Error Correction (SEC) Decoder Overview

As discussed in the previous sections, the coding layer uses a BCH (63,56) Decoder that operates in single error correction and double error detection mode (SEC). This decoder has been specifically selected for the decoding process because this paper assumes that the data on the sending end has been encoded using a BCH (63,56) Encoder, with n = 63 representing the code word length, n – k = 7 parity bits, and k = 56 information bits [2]. This encoding procedure processes 56 information bits and generates 7 parity check bits at the sender's end using the polynomial generator in equation (1) [4], which is used for error control at the receiving end. The data is structured in the format shown in Figure 4 for each BCH code block.

- $g(x) = x^7 + x^6 + x^2 + 1$ Equation (1)

Paper [2] clearly explains the encoding algorithm for the BCH (63,56) code. With this foundational knowledge of the BCH Encoder, we now proceed to the decoding phase, focusing on how the transmitted BCH code block is received and decoded using the SEC Decoder. Assume the transmitted code block is C(x), and due to channel errors e(x), the received code block at the decoder is R(x)

- $R(x)= C(x) + e(x)$ Equation (2)

As mentioned previously, the goal of the decoder is to check the error in the BCH code blocks using SEC mode which can correct single error bit and detect two error bits.

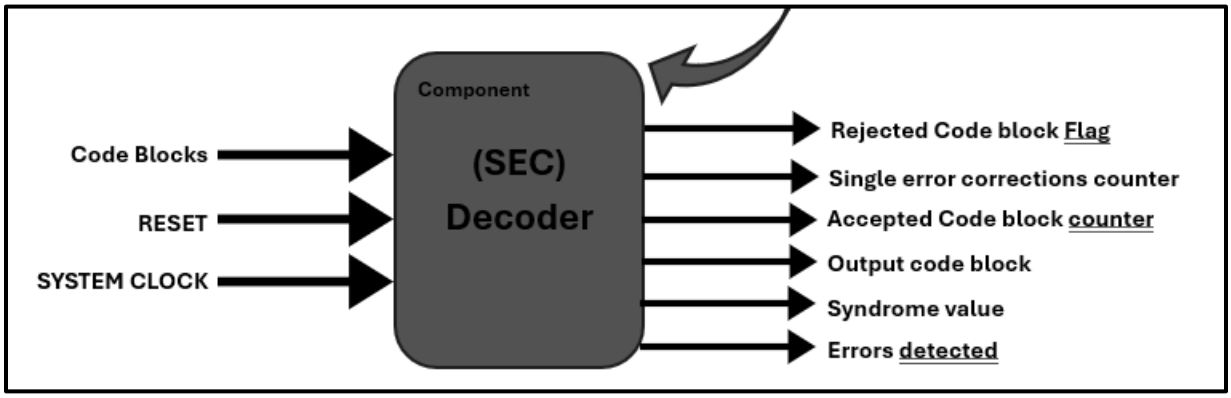



*Figure 24. SEC Decoder Block Diagram*

The BCH decoder operating in SEC mode consists of several functions that generate the outputs demonstrated in the block diagram above. These outputs are used for making final decisions during the decoding process on whether to accept the code block or reject it. The decoder can correct only single-bit errors; if a two-bit error occurs, the code block will be rejected. Below is a description of the output signals for the intended future implementation design of the decoder:

- **Syndrome Value:** The syndrome value is generated by a syndrome generator implemented within the decoder. This generator computes the syndrome by performing a mathematical operation on the received code block [6]. The BCH decoder's syndrome detection circuit can be designed in two ways:

1. Using a syndrome generation (dividing) circuit to calculate the syndrome value and a lookup table to determine the error location based on the calculated syndrome values, as described in [2].

2. Following the SEC mode shift register decoder implementation from [1] & [6].

In both approaches, the decoder takes the 63 bits (56 information bits + 7 parity bits), ignoring the filler bit, to generate the syndrome values:

- If the syndrome binary value is zero (all syndrome bits are zero), the code block is clean and error-free.
- If the syndrome value is non-zero, the decoder attempts to locate the error. If it is a single-bit error, the error is corrected. If more than one bit is erroneous, the block remains uncorrected.

Both approaches will yield the same result: single-bit errors are corrected, and two-bit errors are detected. The difference between [2] and [1],[6] lies in the syndrome function used to generate the values.

- **Code Block Rejection Flag**: This flag acts as an authentication pulse that validates whether the code block is accepted or rejected. If the syndrome value is zero, the authentication pulse goes high, indicating the code block is error-free. If the syndrome value is non-zero, two cases arise:
  - If the code block contains uncorrectable errors (more than one bit error), the authentication pulse is low, marking it as a "dirty" code block.
  - If the error is a correctable single-bit error, the error is corrected, and the authentication pulse goes high.

- **Errors Detected**: This signal specifies whether the error is even or odd.

- **Output:** The output is either the accepted or rejected code block, where the authentication pulse indicates whether the block is valid or dirty.

### 1.6.1 Single Error Correction (SEC) Decoder Operation Logic Flowchart and Decoding Strategy

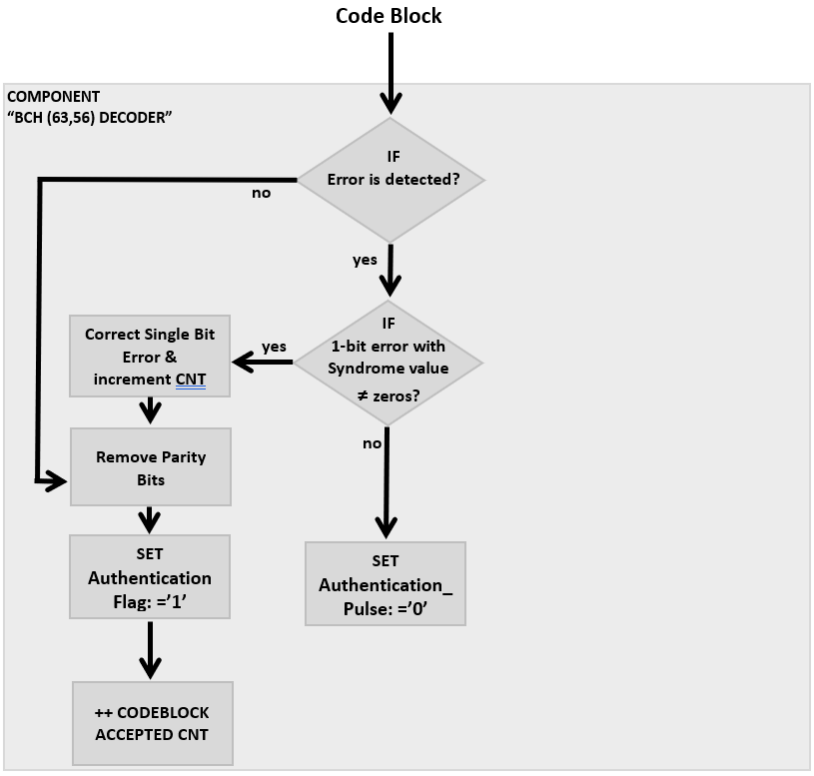



*Figure 25. SEC Decoder Logic Flowchart*

Table 3. SEC Decoding Process Strategy

| Scenario | Syndrome Bit | Error Type | Code block rejection Flag (Authentication pulse) | Description |
|---|---|---|---|---|
| Error-free | = 0 | - | 1 (False) | Valid |
| Erroneous | =/ 0 | 1 -bit error | 1 (False) | Valid |
| Erroneous | =/ 0 | 2-bit error | 0 (True) | Not Valid |

As previously mentioned, missing a Tail Sequence can have significant consequences, as it may prevent the CLTU reception procedure from returning to the SEARCH state for the next CLTU, potentially causing the entire subsequent CLTU to be missed [1]. The SEC decoding process discussed earlier ignores the filler bit during its operation. To reduce the chances of missing a tail sequence, the filler bit will be used as the final determining factor in the decoding process, deciding whether to accept or reject the code block before passing it to the upper layers. This leads to the future implementation outlined in Section 1.7, where we will describe how this decoder will be integrated into the VHDL design we have modeled for the channel coding layer.

### 1.7 Future Work - Upcoming Implementation and Expansion

The purpose of this paper is not limited to implementing the proposed model for the state control logic (FSM) of the channel coding layer using VHSIC Hardware Description Language (VHDL). The goal is to expand and enhance the design, eventually completing the entire channel coding layer. Below are the planned areas for further study and implementation to improve the performance of the current design. The approach is to start small and continually build upon and refine the code. The following points outline the future work that will continue from this paper:

- Enhancing the CLTU reception procedure: The implemented VHDL design will be improved for detecting the start sequence in relation to error handling. This will include the detection of both the start sequence (0xEB90) and its inverse (0x146F), which addresses data ambiguity in the received bit stream caused by non-return-to-zero (NRZ) encoding in the incoming start bit streams with allowing single bit error anywhere.

- Testing with consecutive CLTUs: The VHDL code was initially tested with single CLTUs sent independently (where TCA's drop between CLTUs, and the entire resynchronization process begins again). The future plan is to test the design's performance when sending consecutive CLTUs while maintaining bit synchronization on the channel. This also includes evaluating whether CLTU organization could be improved by inserting idle or tail sequences between them. This extension will be designed in VHDL and added to the current implementation [5].

- Developing a VHDL code for the SEC decoder: Further research will be conducted to develop the SEC decoder following [1] and [6] 's approach. This implementation will include inverters, several shift registers, buffer registers for temporarily storing information and parity bits, an EOD (Even-Odd Detector) to check parity, an SR (Syndrome Register) to calculate the syndrome, and a PLR (Position Location Register) to detect errors within the code block. A design study will be carried out to implement this system as recommended by CCSDS. The figure below shows the circuit implementation of the Error Correction Mode Decoder provided by CCSDS [1].

- Incorporating the filler bit into the SEC decoder: The filler bit will be considered in the final decoding strategy for improved performance. The intended algorithm for the decoding process at the receiving end is as follows:
    1. Test the received code block (excluding the filler bit) for errors using the SEC.
    2. If no errors are detected, the code block is accepted and sent to the upper layer without checking the filler bit.
    3. If one error is detected, check the filler bit:
        - (a) If the filler bit is 0, the code block is corrected and accepted.
        - (b) If the filler bit is 1, a CODEBLOCK REJECTION is declared.
    4. If two errors are detected, the code block is rejected, and the filler bit is ignored.

    However, further research is needed to develop and implement this in the design.

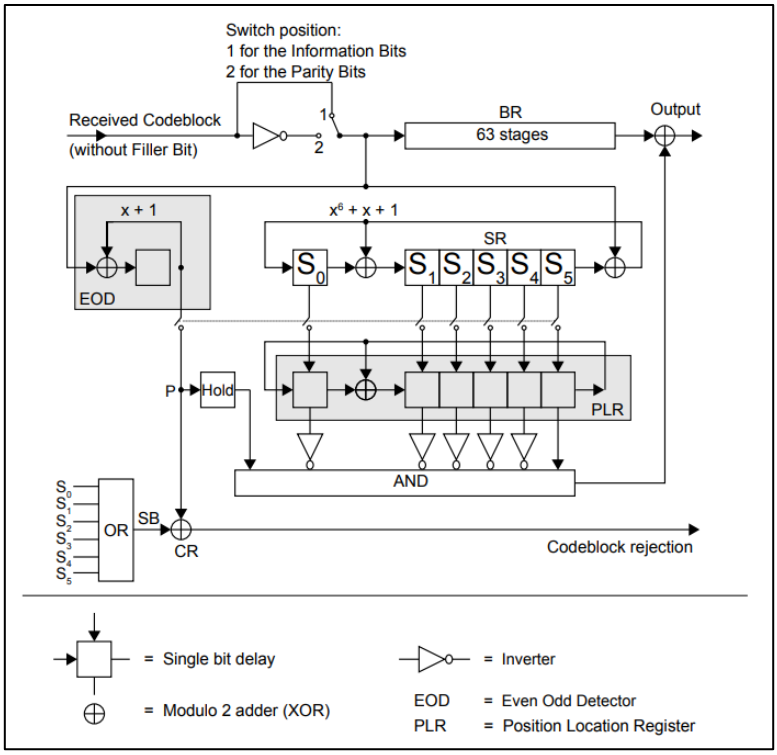



*Figure 26. SEC Decoder Operation [1]*

### 1.8 Conclusion

The Channel Coding Layer is a critical layer of the telecommand subsystem in any communication satellite. Following the recommendations of CCSDS, we successfully designed a VHDL code for the state control logic at the receiving end, which accomplished essential tasks such as input synchronization, channel activation detection, start sequence detection, data stream sampling, and channel selection. These features, along with the critical events outlined in Section 1.4, ensure the design meets the performance requirements according to CCSDS standards.

While the SEC decoder was treated as a black box in this initial VHDL implementation, it will be fully integrated in future work, as detailed in Section 1.7. This addition will expand the current design, providing error correction capabilities and further enhancing the robustness of the implementation. Additionally, several other improvements are planned to elevate the performance and functionality of the coding layer, positioning this work as a foundational step toward a more sophisticated and reliable coding layer for a packet telecommand decoder system.

## REFERENCES


[1] Consultative Committee for Space Data Systems. *TC Synchronization and Channel Coding—Summary of Concept and Rationale*. Issue 3, CCSDS 230.1-G-3, October 2021, Washington, DC, CCSDS.

[2] Arunkumar, S., and T. Kalaivani. *FPGA Implementation of CCSDS BCH(63,56) for Satellite Communication*. IEEE International Conference on Electronics Design, Systems and Applications, 2012.

[3] Consultative Committee for Space Data Systems. *TC Synchronization and Channel Coding*. Issue 4, CCSDS 231.0-B-4, July 2021, Washington, DC, CCSDS.

[4] *MA28140: Packet Telecommand Decoder Datasheet*. July 2002.

[5] Consultative Committee for Space Data Systems. *Telecommand: Summary of Concept and Service*. CCSDS 200.0-G-6, Issue 6, January 1987, Washington, DC, CCSDS.

[6] Nugroho, M., and I. Choiriyah. *Simulation of coding layer of telecommand based on the Consultative Committee for Space Data Systems recommendation*. IOP Conference Series: Earth and Environmental Science, vol. 284, 2019, p. 012050.

*[7] Mathew, Priya, Lismi Augustine, Sabarinath G., and Tomson Devis.* Hardware Implementation of (63, 51) BCH Encoder and Decoder for WBAN Using LFSR and BMA*. Department of ECE, St. Joseph's College of Engineering & Technology, Palai, Kerala.*