3.0 YARR Tx/Rx

3.1 Data Format and Protocol

To understand the SEE detection and recovery system, an understanding of the logic and data surrounding the transmitting end of the RD53 readout chip and receiving end of a DAQ is necessary. At the highest level, the connection between the YARR DAQ and RD53 pixel readout chip is made up of one or multiple differential serial data connections. For simplicity, a simple case with just a single differential data line will be considered and is shown in Figure X. This connection has no control, handshaking, or reference clock signals but purely a single serial wire transmitting a bit at a time. Communicating serially without these additional signals poses several analog and logical challenges.

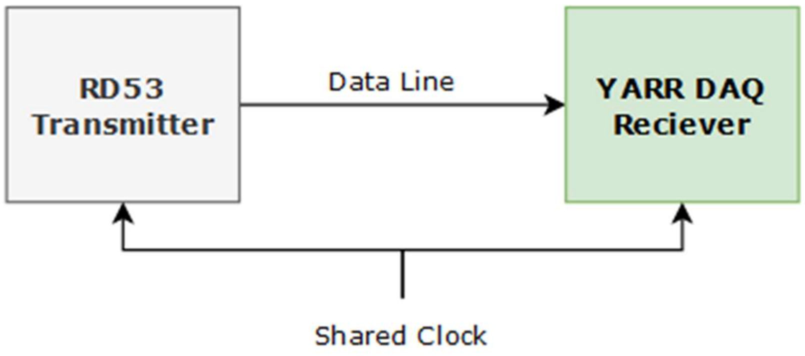

Figure X: [Description of figure]

To address these issues, communication uses a custom 64b/66b line encoding. *[A line to introduce the goals goes here]*

- **Clock recovery**. Clock recovery utilizes incoming data transitions to allow a receiver to determine the timing of incoming data without a separate clock. This feature was relevant previously isn't utilized by the YARR and RD53.
- **Steam Alignment**. Stream alignment allows a receiver to determine packet boundaries in a continuous stream of data. This is accomplished by the receiver's synchronization scheme.
- **DC Balance**. DC balance standards require an even number of 1s and 0s in a transmission. This mitigates the biasing of voltage which has adverse effects on level detection theshhold circuitry. *[This is accomplished by the transmitter's scrambler.]*

- **Transition Density**. Transition density is the ratio of 0 to 1 or 1 to 0 transitions within a stream to the number of bits transmitted. *[This is accomplished by the transmitter's scrambler.]*
- **Run Length**. Run length is the maximum consecutive 0s or consecutive 1s in a stream. Although 66b/64b encoding doesn't guarantee a maximum it imposes statistical bounds. *[This is accomplished by the transmitter's scrambler.]*

In 64b/66b encoding, data blocks are 66 bits and have two distinct elements. The two most significant bits in the blocks are the header, which is strictly a 01 or 10. This sets a maximum upper bound on run length and guarantees that one transition will occur every 66 bits. The remaining 64 bits are scrambled data. The data needs to be unscrambled on the receiver end to recover the original 64 bit payload. Xilinx provides an IP core called 'Aurora 66B64B', however, it is not compatible with all Xilinx FPGA families and utilizes specialized hard silicon as well as a lot of additional logical overhead to interface it. Instead, a custom protocol is used which utilizes more basic and accessible primitives while keeping the block codes the same as that of the Xilinx implementation [citation].

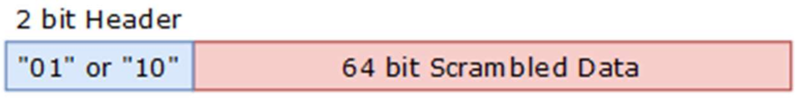

Figure X: [description of image]

3.2 Pixel Readout Tx Logic

The pixel readout chip can have multiple simultaneous data output streams; however, each stream utilizes identical logic to package and transmit the data. Each port is made up of 3 major blocks: the scrambler, the gearbox, and serializer. These transform the data from a parallel framed 66-bit block in the format given above, to a scrambled block serialized at high speeds across the communication link.

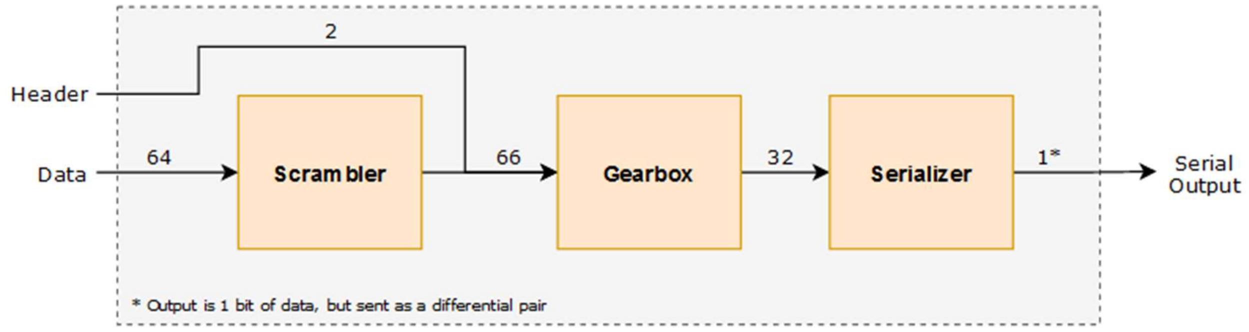

Figure X: [description of image]

3.2.1 Tx Scrambler

To accomplish the DC balancing, transition density, and run length requirements set by the line encoding algorithm, the data being transmitted must be modified. Data stored in registers, and data produced from atomic collisions doesn't tend to already have the characteristics required. To transform it so that it does, the data is passed through the scrambler. The scrambler takes a block of a set bit width and applies a function to it to produce a second block that is statistically likely to have a short run length, a high transition density and an equal number of 1s and 0s.

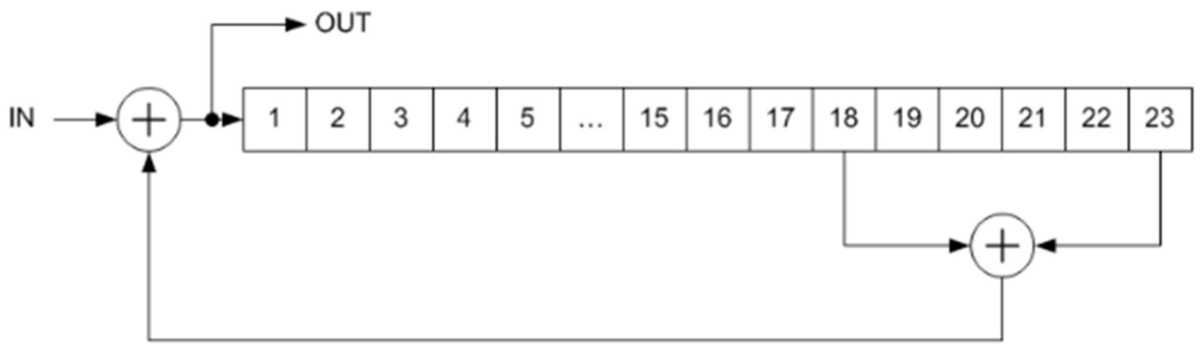

Figure X: [description] [citation]

Scramblers are described with a polynomial which tells you the tap values to use for feedback. Figure X shows an example scrambler with the function $1 + x^{18} + x^{23}$. While the scrambler looks like a serial operation, it can, with sufficient resources and time, evaluate a full 64-bit block simultaneously. One thing to note is that the operation has memory so that the generation of bits for the output depends on the previously generated bits. Feeding the scrambler the same block continuously (for example an IDLE) will produce different outputs cycle by cycle.

The scrambler used by the RD53 pixel readout chip uses a $1 + x^{39} + x^{58}$ polynomial as per Xilinx documentation for its own IP 64b/66b protocol [citation]. This scrambler is multiplicative self-synchronizing, meaning that this transmission scrambler and its receiver counterpart can be initiated at different times or with different states but still achieve synchronization. The scrambler in the RD53 receives a 64 block of payload data, which it transforms into 64 bits of scrambled data. This data is appended with two header bits (which are not scrambled) and passed forward to the gearbox.

Before moving to the gearbox, it may seem like scrambling the data to introduce more transitions will work counter to the stream alignment requirement. With more transitions, how can the receiver be certain which transition is indicative of the header and which is simply well-scrambled data? However, since the scrambler produces essentially randomized data, each pair of bits, except the designated header, in the stream is 50% likely to not contain a transition. Therefore, if we watch a position for *n* blocks, there is only $\frac{1}{2^n}$ probability of having a transition appear in all *n* blocks, unless of course it is the correct header.

3.2.2 Tx Gearbox

Between the scrambler and the serializer is the gearbox. The gearbox receives the 2 bits of header and 64 bits of scrambled data and packages them into 32-bit chunks to prepare them for the 32-bit serializer. The data is shifted out of the gearbox at twice the clock rate as it is shifted in. Therefore, every input clock cycle, 66 bits are shifted into the gearbox and 64 bits are shifted out. A result of this input output size mismatch is that there is an accumulation of 2 bits every input cycle. To address this there is a pause every 33rd cycle to shift out the accumulated bits and allow the gearbox to “catch up”.

[simple image depicting the gearbox – may not be necessary]

3.2.3 Tx Serializer

The final step in the output sequence is the serializer. The serializer shifts in 32 bit chunks and shifts out each serially, meaning a single bit at time. To accomplish this the serializer uses a faster, clock as well as DDR data transmission to transfer data on both the rising and falling edges of the clock. The abstract behavior can be described as a parallel in, serial out register with a bit width of 32. Serializer parameters are given below.

Property	***Setting***

Bandwidth	1.28 Gbps
Interface Template	Custom
Data Bus Direction	Output
Data Rate	DDR (Dual Data Rate)
Serialization Factor	8
External Data Width	1
I/O Signaling	Differential (LVDS)
clk_in (freq)	640 MHz
clk_div_in (freq)	160 MHz

Table X: [caption][citation]

3.3 DAQ Rx Logic

The Rx logic within the DAQ mirrors that of the pixel redout Tx. Each of the blocks used to implement the line encoding have counterparts to undo the encoding in the reverse order. Built into each block as well as some periphery, however, is additional logic to ensure that stream alignment is maintained. Just like with the Tx, multiple duplicate channels can simultaneously service multiple input lines; however, for simplicity we consider just a single channel.

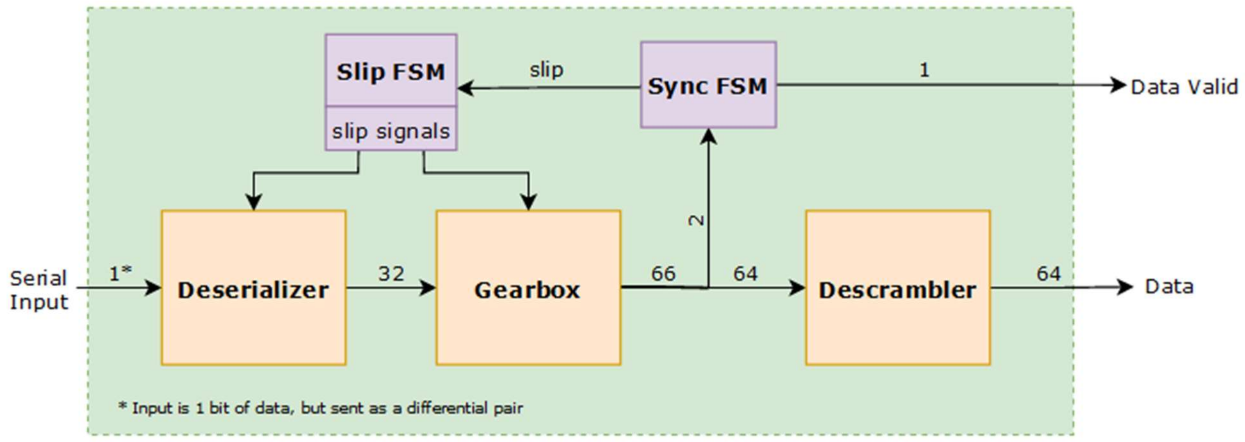

Figure X: Description of Image

3.3.1 Rx Deserializer

The receiving end deserializer is functionally very similar to the transmission serializer except that it performs the inverse operation. The deserializer receives serial data at a high rate which is accumulated into a 32-bit chunk and passed inward to the gearbox. Just like the serializer, the deserializer can be modeled as a serial in parallel out register. The data is grouped into four 8-bit sets. These sets are then assembled into a 32-bit chunk. Just as in the Tx serializer, the deserializer's parameters are given below:

Property	***Setting***
Bandwidth	1.28 Gbps
Interface Template	Custom
Data Bus Direction	Input
Data Rate	DDR (Dual Data Rate)
Serialization Factor	8
External Data Width	1
I/O Signaling	Differential (LVDS)
clk_in (freq)	640 MHz
clk_div_in (freq)	160 MHz

Table X: [caption][citation]

3.3.2 Rx Gearbox

Just like in the transmitter, the receiver gearbox's purpose is to perform a transformation between 32-bit chunks from the deserializer, back into the 66-bit RD53B block. In this case the reverse of the Tx side is accomplished. The gearbox shifts in two 32-bit chunks and shifts out a single 66-bit block along with a block valid signal every cycle. Although the behavior is similar to the Tx gearbox, understanding the Rx gearbox is vital to understanding the resynchronization scheme in place as well as the proposed replacement.

The basic behavior of the gearbox is to shift stream data through a buffer. The deserializer produces and passes 32-bit chunks to the gearbox, where it is shifted into an internal 128-bit buffer every half cycle. On the output, a 66-bit block is shifted out every cycle, containing the original scrambled data and unscrambled header. Because of this 64-to-66-bit transform, 2 bits more are taken out of the buffer every cycle than is shifted in.

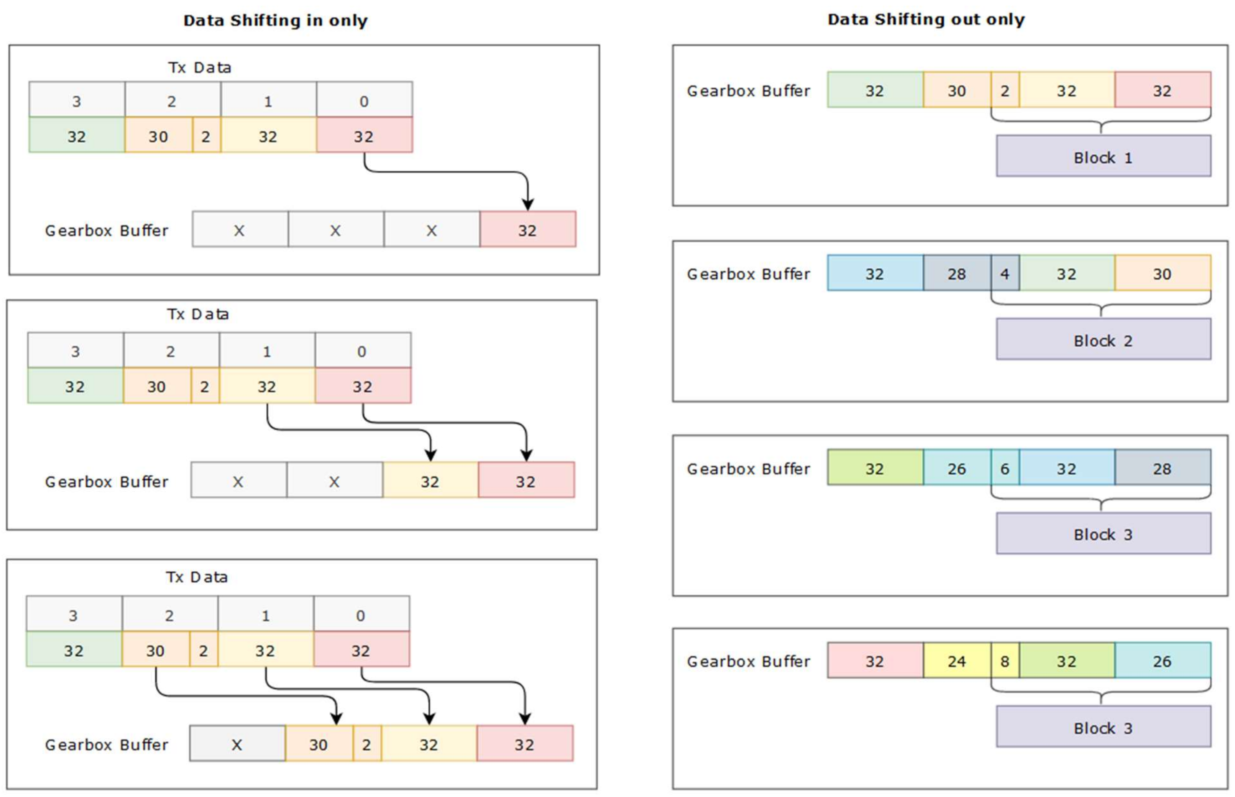

Figure X: [Describe image]

The 66-bit block on the output of the gearbox is selected by a sliding window. While the buffer is 128-bits wide, a select 66 bits are chosen every cycle for the output word. In the first cycle, bits 127 down to 62 of the buffer are chosen for the first 66-bit block. In the second cycle, bits 125 down to 60 are read out for the second 66-bit block. This pattern continues until bits 65 down to 0 are read out. The reason for this sliding window is because the gearbox has to make up for the 2 extra bits that are shifted out from the buffer than shifted in by adjusting the expected position of the next block.

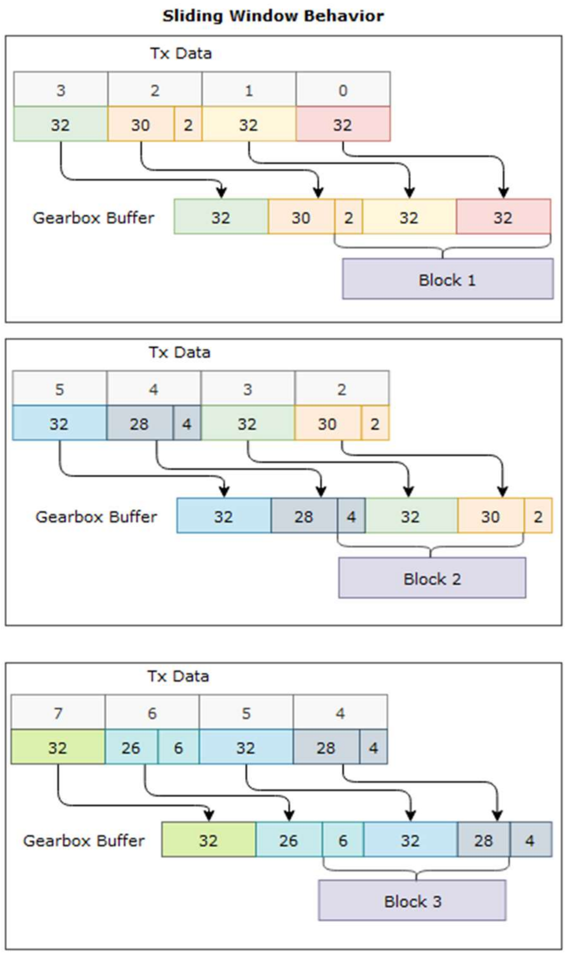

Figure X: [Describe Image]

Every 33rd cycle there is a pause, and no 66-bit block is produced. Once the sliding window reaches bits 65 down to 0, there is no further it can go, and the gearbox has to wait for the buffer to fill up again through data being shifted in from the serializer. After the pause, the sliding window is reset to its initial position at bits 127 down to 62, where another 32 66-bit blocks are produced before the next pause.

3.3.3 Rx Descrambler

Of the 66-bit block produced by the gearbox, the two most significant bits (the header) are checked for validity, while the remaining 64 scrambled bits are passed through the descrambler. The Rx descrambler is the reverse of the Tx scrambler and therefore has identical properties to it. It is multiplicative and self-synchronizing and utilizes the same polynomial, $1 + x^{39} + x^{58}$, to derive the unscrambled data. A smaller example descrambler is shown in the figure below.

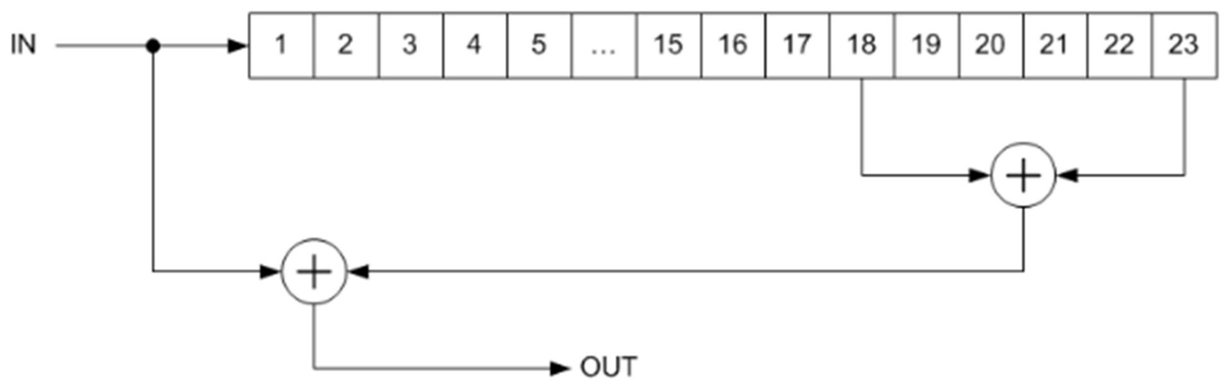

Figure X: [description] [citation]

4.0 YARR’s Original Detection and Resynchronization Scheme

Built into the Rx logic of the YARR is a set of logic to verify and maintain stream alignment. Both the RD53 Pixel Readout and YARR send and receive data using the same shared clock. Communication between the two takes a fixed but unknown delay, but since the clock is shared the sending and sampling is expected to happen at the same rate. All that the YARR receiver must do then is search and find the packet boundary in the incoming stream. This boundary is set by the header bits and are differentiated from the remaining data by always containing a 0 to 1 or 1 to 0 transition. Since blocks are 66 bits wide, the receiver checks a pair of bits periodically for every 66 streamed through to ensure that the same bit indices of each block are checked. If the bits checked consistently contain a transition, then the boundary is found, the sender and receiver are synchronized, and the receiver will correctly distinguish blocks within the stream. However, if the bits checked in some blocks do not contain a transition, i.e. they are from the data field rather than the header, then the YARR needs to change the block bit indices it is periodically checking to search elsewhere. This is accomplished by shifting the stream data, effectively dropping or duplicating a bit in the stream, so as to change the indices being periodically checked. With enough shifts the YARR will be able to scan through all 66 possible bit indices.

In order to shift the stream data, the Rx logic utilizes a control logic FSM to allocate the shifts of the stream by leveraging features of the deserializer and the gearbox. The FSM also determines when synchronization is obtained or lost.

4.1 Serializer Bitslip

The deserializer in the YARR DAQ is made up of a series of Xilinx primitives, including the ISERDESE2. Built into this primitive is a “bitslip module” whose original purpose was to lock onto a repeating pattern, called a training pattern, to ensure that the deserializer was ordering the received bits in the correct orientation. Since the slip effectively drops bits, the alignment control unit leverages this feature to shift the position in which the header bits appear. To keep up with the data rate required, the deserializer is operated in DDR mode, which has a somewhat complex bit slipping behavior. The bit slipping model can be depicted as a 15-bit buffer from which an 8 bit window decides the data that is read out.

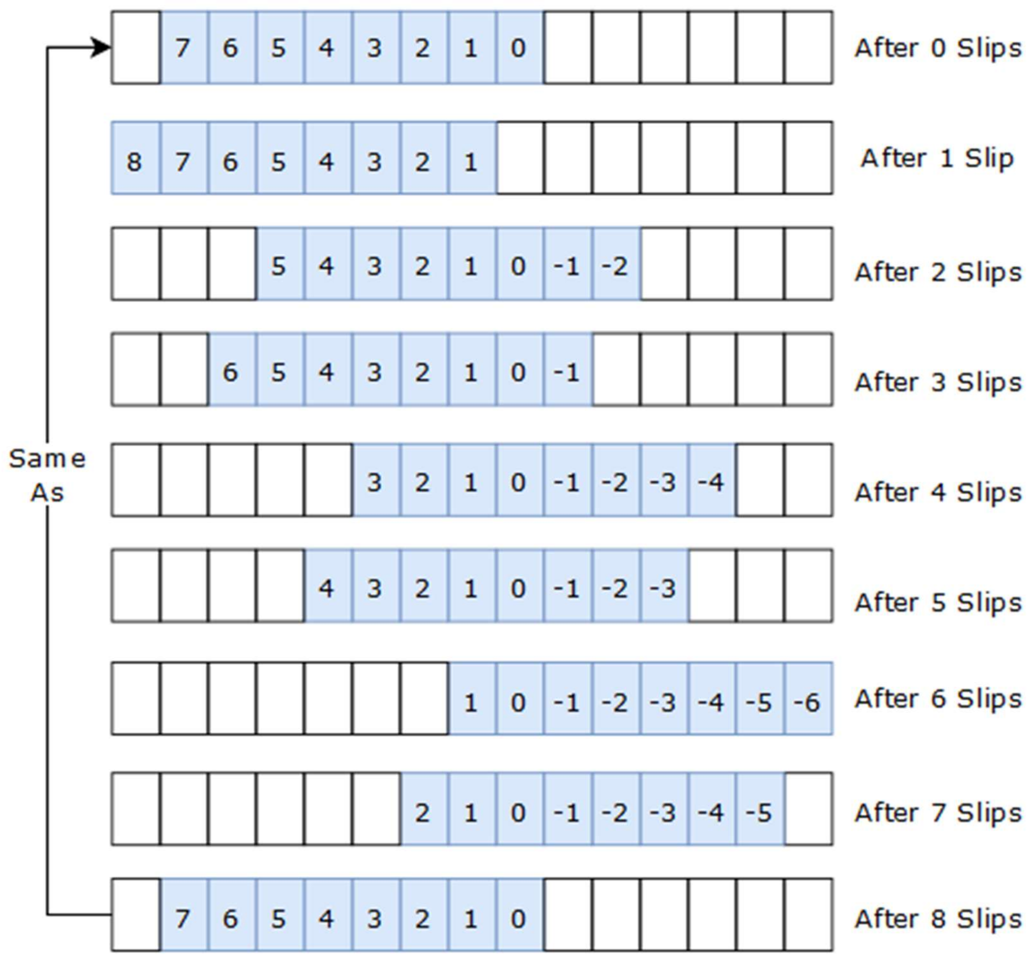

Figure X: Describe the image

Since the buffer available to the deserializer is limited in size, at most 7 bits can be dropped, and 8 header positions can be evaluated. Once all 8 are seen, the deserializer snaps back into its original position. Despite the back and forth slipping behavior of the DDR bit slip, the bit slipping generally shifts the data stream in a single direction, which, as we will show later, results in alignment being recovered more quickly for SEE induced stream bit drops over stream bit adds.

4.2 Gearbox Slip

8 bits of range is inadequate with block sizes of 66 bits, and so a second mechanism for shifting the data stream is required. The gearbox fills this role by taking advantage of its larger buffer where it can search a much greater range compared to the deserializer. The gearbox uses a sliding window which it uses to select 66 bits out of its buffer to produce a block. By allowing the alignment control FSM to actively move this window, the entire 128-bit buffer can be scanned