6/29 UPDATES:

1. The PositionRead and StatusRead signals are not asserted until AFTER the second SynchedTick pulse is sent.

2. The PositionRead and StatusRead signals are now staggered by one 60MHz clock cycle, they should not happen at the same time.

3. Pushed out the second SynchedTick pulse to 30 us.

To understand the signals' role in setting up the conditions required for proper initialization of **SSI\_CLK**, let's go through each signal and its relationship to **SSI\_CLK**:

1. **ClkOn**: This signal enables the external SCLK functions. It must be asserted for the clock signal generation to occur.
2. **ShiftOn**: This signal enables the shifting of external data into a holding register. It must be asserted for the data to be shifted in during the clock signal generation.
3. **TurnShiftOff**: This signal indicates when the shifting process should be turned off. When asserted, it stops the shifting of data.
4. **PreTurnShiftOff**: This signal determines when the **TurnShiftOff** signal will be asserted. If this signal is asserted, **TurnShiftOff** will be asserted after the current shift operation is complete.
5. **intSSI\_CLK**: This is the clock generated for the SSI data transfer. It toggles at twice the frequency of the desired **SSI\_CLK** output signal.
6. **ToggleEn**: This signal must run at twice the frequency of **SSI\_CLK** and is used to toggle the **SSI\_CLK** output. When asserted, **SSI\_CLK** will toggle.
7. **Enable**: This signal indicates whether the SSI controller is enabled. If not enabled, the sequence is considered over, and the **CycleCounter** will be reset to zero.
8. **SequenceOn**: This signal determines whether the SSI read sequence is active. It is active when **SequenceOn** is asserted and **CheckDataLo** is deasserted.
9. **CycleCounter**: This signal represents the current count of the SSI read sequence. It increments when **Enable** is asserted.
10. **HalfPeriod**: This signal determines the value at which **CycleCounter** should match for **CycleCountMatch** to be asserted. It represents half of the desired period for **SSI\_CLK**.
11. **CycleCountMatch**: This signal indicates when the **CycleCounter** matches the **HalfPeriod** value. It serves as a condition for toggling **ToggleEn** and generating the clock signal.
12. **ShiftCounter**: This signal counts the number of bits that have been shifted in during the SSI read cycle.
13. **StartRead**: This signal is a pulse that initiates the SSI read cycle when asserted.
14. **CheckDataHi**: This signal is used to check and latch the data line status before data transfer.
15. **CheckDataLo**: This signal is used to check and latch the data line status after data transfer.
16. **CheckDataDelay**: This signal introduces a delay before sampling the line break detection bit. It ensures that the data line has enough time to stabilize before checking for a line break.

The correct initialization and coordination of these signals are crucial for generating the desired **SSI\_CLK** waveform. Issues such as **SSI\_CLK** not oscillating correctly may arise if these signals are not properly synchronized or configured. It's essential to review the interdependencies among these signals and ensure they are correctly set and coordinated to achieve the desired behavior of **SSI\_CLK**.

1. **ClkOn**: This signal must be asserted (**'1'**) to enable the external SCLK functions. It allows the clock signal generation to occur.
2. **ShiftOn**: This signal must be asserted (**'1'**) to enable the shifting of external data into a holding register during the clock signal generation.
3. **TurnShiftOff**: This signal should be deasserted (**'0'**) to keep the shifting process active. When asserted (**'1'**), it stops the shifting of data.
4. **PreTurnShiftOff**: This signal does not directly affect the state of **SSI\_CLK** but determines when the **TurnShiftOff** signal will be asserted. If **PreTurnShiftOff** is asserted (**'1'**), it indicates that **TurnShiftOff** will be asserted after the current shift operation is complete.
5. **intSSI\_CLK**: This clock signal is generated for the SSI data transfer. It should toggle at twice the frequency of the desired **SSI\_CLK** output signal.
6. **ToggleEn**: This signal should run at twice the frequency of **SSI\_CLK** and is used to toggle the **SSI\_CLK** output. It needs to be asserted (**'1'**) to enable the toggling of **SSI\_CLK**.
7. **Enable**: This signal should be asserted (**'1'**) to enable the SSI controller. If it is deasserted (**'0'**), the sequence is considered over, and the **CycleCounter** will be reset to zero.
8. **SequenceOn**: This signal should be asserted (**'1'**) to indicate that the SSI read sequence is active. It should be active when **SequenceOn** is asserted and **CheckDataLo** is deasserted.
9. **CycleCounter**: This signal represents the current count of the SSI read sequence. It should increment when **Enable** is asserted.
10. **HalfPeriod**: This signal represents half of the desired period for **SSI\_CLK**. The **CycleCounter** should match this value for **CycleCountMatch** to be asserted.
11. **CycleCountMatch**: This signal indicates when the **CycleCounter** matches the value of **HalfPeriod**. It serves as a condition for toggling **ToggleEn** and generating the clock signal.
12. **ShiftCounter**: This signal counts the number of bits that have been shifted in during the SSI read cycle. It does not directly affect the state of **SSI\_CLK**.
13. **StartRead**: This signal should be asserted (**'1'**) to initiate the SSI read cycle.
14. **CheckDataHi**: This signal is used to check and latch the data line status before data transfer. It does not directly affect the state of **SSI\_CLK**.
15. **CheckDataLo**: This signal is used to check and latch the data line status after data transfer. It does not directly affect the state of **SSI\_CLK**.
16. **CheckDataDelay**: This signal introduces a delay before sampling the line break detection bit. It does not directly affect the state of **SSI\_CLK**.
17. **ClkOn** must be asserted (**'1'**): This signal enables the external SCLK functions and allows the clock signal generation to occur.
18. **ToggleEn** must be asserted (**'1'**): This signal should run at twice the frequency of **SSI\_CLK** and is used to toggle the **SSI\_CLK** output.
19. **intSSI\_CLK** must be deasserted (**'0'**): This is the clock generated for the SSI data transfer. When it is deasserted, it indicates that the clock is low.
20. **ShiftOn** must not be currently asserted: The **ShiftOn** signal should only be asserted when it is not already in the asserted state.

By meeting these conditions, the **ShiftOn** signal can be properly defined. Specifically, it should be asserted (**'1'**) when **ClkOn** and **ToggleEn** are both asserted, and **intSSI\_CLK** is deasserted (**'0'**). However, it should only be asserted if it is not already in the asserted state, ensuring that it is toggled properly.

1. ToggleEn: ToggleEn needs to be high when ClkOn is high, intSSI\_CLK is low, and ShiftOn is low. Based on the code, ToggleEn is set high when ClkOn, CycleCountMatch, and Enable are all true. So the condition for ToggleEn can be expressed as: ToggleEn <= '1' when ClkOn = '1' and not intSSI\_CLK = '0' and ShiftOn = '0' and CycleCountMatch = '1' and Enable = '1' else '0'.
2. intSSI\_CLK: According to the code, intSSI\_CLK is generated based on the conditions when StartRead is true and Enable is true. intSSI\_CLK is set high by default, but it can be set low in the following condition: intSSI\_CLK <= '0' when StartRead = '1' and Enable = '1' else '1'.
3. ShiftOn: ShiftOn is set high when ClkOn, ToggleEn, and intSSI\_CLK are all true. The condition for ShiftOn can be expressed as: ShiftOn <= '1' when ClkOn = '1' and ToggleEn = '1' and not intSSI\_CLK = '0' else '0'.

Correct conditions for asserting ToggleEn, intSSI\_CLK, and ShiftOn properly:

1. ToggleEn: ToggleEn needs to be high when ClkOn is high, intSSI\_CLK is low, and ShiftOn is low. Based on the code, ToggleEn is set high when ClkOn, CycleCountMatch, and Enable are all true. So the condition for ToggleEn can be expressed as: ToggleEn <= '1' when ClkOn = '1' and not intSSI\_CLK = '0' and ShiftOn = '0' and CycleCountMatch = '1' and Enable = '1' else '0'.
2. intSSI\_CLK: According to the code, intSSI\_CLK is generated based on the conditions when StartRead is true and Enable is true. intSSI\_CLK is set high by default, but it can be set low in the following condition: intSSI\_CLK <= '0' when StartRead = '1' and Enable = '1' else '1'.
3. ShiftOn: ShiftOn is set high when ClkOn, ToggleEn, and intSSI\_CLK are all true. The condition for ShiftOn can be expressed as: ShiftOn <= '1' when ClkOn = '1' and ToggleEn = '1' and not intSSI\_CLK = '0' else '0'.

\*\*Primary Signals and Functionality:\*\*

1. \*\*SSIDataLatch\*\*: It's an internal register storing data received from the SSI interface. This value changes during the process driven by the SysClk signal when the SynchedTick signal is high. The content of SSIDataLatch is a dynamic value that is updated with the value of Serial2ParallelData on each SynchedTick pulse.

2. \*\*ssiDataOut\*\*: The output of the module can output either the SSIDataLatch contents or status information, depending on the values of PositionRead, SSISelect, and StatusRead.

- When PositionRead and SSISelect are high, it outputs the content of SSIDataLatch.

- When StatusRead and SSISelect are high, it outputs status information, which includes whether there's no transducer connected (NoXducer), whether the data is valid (DataValid), the clock rate, data length, and transducer select settings.

- If neither (PositionRead and SSISelect) nor (StatusRead and SSISelect) are high, it outputs an all-zero vector.

3. \*\*Status Information\*\*: This reflects the configuration we have set (ClockRate, DataLength, TransducerSelect, etc.) as well as the NoXducer and DataValid flags.

4. \*\*DataValid\*\*: The DataValid signal is set to '1' after a successful data read and remains so until the PositionRead signal falls.

5. \*\*SynchedTick pulses\*\*: The SynchedTick generates two pulses at 30 MHz as per the provided intervals. The first pulse occurs at 2 microseconds and the second pulse at 10 microseconds. These pulses are essential for controlling the timing of various operations.

6. \*\*Wire Break Detection\*\*: The Wire Break detection functionality checks if the SSI\_DATA line is not high at the start or not low at the end of the read cycle. A wire break is detected, and the corresponding signals DataLineHi and DataLineLo should reflect this.

\*\*SSI Transducer Output based on Signals:\*\*

1. \*\*ssiDataOut\*\*: Depends on PositionRead or StatusRead is active and if SSISelect is high.

- If PositionRead and SSISelect are high, the value on ssiDataOut will equal the value on SSIDataLatch.

- If StatusRead and SSISelect are high, ssiDataOut will be a 32-bit value composed of several fields (including NoXducer, DataValid, ClockRate, DataLength, TransducerSelect).

- If neither PositionRead nor StatusRead are high (or SSISelect is low), ssiDataOut will be "0000\_0000".

2. \*\*SSI\_CLK\*\*: This is the clock signal for the SSI data transfer. It gets toggled every time ToggleEn and ClkOn are both high, otherwise it remains at its last value. It's also set to high during the StartRead process.

3. \*\*SSISelect\*\*: This will be high when the transducer selected is either SSIBinaryAnalog or SSIGrayAnalog, otherwise it will be low.

4. \*\*NoXducer\*\*: It will be high if there is a break in the wire or connection to the transducer (DataLineHi or DataLineLo is high), otherwise it will be low.

These values are all subject to change based on the input signals and the internal processes that run when the H1\_CLKWR or SysClk signals have a rising edge.

intSSI\_CLK signal is driven by the following condition: ToggleEn and ClkOn. Let's analyze these signals further:

ToggleEn: This signal determines whether the clock toggling should occur. It is derived from the condition CycleCountMatch and Enable. CycleCountMatch is high when the CycleCounter matches the HalfPeriod value and Enable is the periodic enable signal.

ClkOn: This signal enables the external SCLK functions and is active during the SSI read cycle. It is generated based on the condition SequenceOn and (not CheckDataLo). SequenceOn is high when the SSI read cycle is started (StartRead), and CheckDataLo is low, indicating that the previous SSI read cycle has ended.

Given this analysis, the possible reasons why SSI\_CLK may not be defined correctly are:

* The CycleCounter does not match the HalfPeriod value, resulting in ToggleEn being low and SSI\_CLK not toggling.
* The SequenceOn signal is not active during the SSI read cycle, leading to ClkOn being low and SSI\_CLK not toggling.
* To troubleshoot the issue, you should examine the values of CycleCounter, HalfPeriod, SequenceOn, and CheckDataLo during the simulation. Verify if they are transitioning as expected and meeting the necessary conditions for SSI\_CLK toggling.
* You can add print statements or waveform visualization to monitor these signals and identify any discrepancies. Additionally, check the timing and synchronization between different processes and signals in the design to ensure correct operation.
* By analyzing the values and behavior of these signals in conjunction with the source code, you should be able to determine why SSI\_CLK is not defined properly and make the necessary adjustments to resolve the issue.

For the SSI\_CLK signal to become defined in this code, several conditions need to be met, as it's directly driven by the internal signal **intSSI\_CLK** and is updated in the main **SysClk**-based process. Here are the conditions:

1. **SysClk must have a rising edge**: **intSSI\_CLK** is only updated in the process block that is triggered on the rising edge of **SysClk**.
2. **StartRead must be '1' and Enable must be '1'**: On the start of a read cycle (**StartRead = '1'**) and when the system is enabled (**Enable = '1'**), **intSSI\_CLK** is set to '1'.
3. **ClkOn must be '1' and ToggleEn must be '1'**: When **ClkOn** and **ToggleEn** are both '1', **intSSI\_CLK** is toggled. **ClkOn** represents whether the external SCLK functions are enabled, and **ToggleEn** controls the frequency of the SSI clock.

Now, the conditions for **ClkOn** and **ToggleEn** to be '1' are as follows:

* + For **ClkOn** to be '1':
    - The condition **StartRead** must be '1'.
    - Or, the condition **ClkOn and not TurnShiftOff** must be met.
  + For **ToggleEn** to be '1':
    - **ClkOn** and **CycleCountMatch** must be '1' and **Enable** must be '1'.

Now, let's break down these conditions further:

* **StartRead condition**: StartRead becomes '1' when **DelayCounter** equals **DelayTerminalCount**, i.e., the delay counter has expired. The DelayCounter increments with each rising edge of **SysClk** when **DelayCntEn** is '1' and **SlowEnable** is '1'. **DelayCntEn** becomes '1' when **SynchedTick** and **SSISelect** are '1' and becomes '0' when **StartRead** is '1'.
* **Enable condition**: **Enable** is a system input signal that is used directly in the main process. It's part of the SSITop module's input port.
* **TurnShiftOff condition**: **TurnShiftOff** becomes '1' when **DataLength** equals **ShiftCounter**. **ShiftCounter** increments each time a shift operation occurs, which is when **ToggleEn and not intSSI\_CLK and ShiftOn** is '1'. **ShiftOn** is set to '1' when **ClkOn and not ShiftOn and not intSSI\_CLK and ToggleEn** is '1' and is set to '0' when **TurnShiftOff** is '1'.
* **CycleCountMatch condition**: **CycleCountMatch** becomes '1' when **CycleCounter** equals **HalfPeriod**. **CycleCounter** increments when **Enable** is '1' and resets when **not SequenceOn or ToggleEn** is '1'.
* **SSISelect condition**: **SSISelect** is '1' when **TransducerSelect(3 downto 0)** equals either **SSIBinaryAnalogXducer(3 downto 0)** or **SSIGrayAnalogXducer(3 downto 0)**. The **TransducerSelect** signal gets its value from the input **intData** during a write operation controlled by the **ParamWrite1** signal.

Conditions for valid data to be read from the ssiDataOut line in the SSITop architecture:

* The PositionRead signal must be active (PositionRead = '1').
* The SSISelect signal must be active (SSISelect = '1'), indicating the correct transducer select.
* The SynchedTick signal must send two 30Mhz sync pulses, one at 2 us and another at 10 us.
* The internal DataValid state (intDataValid) must be true, which is determined by the previous DataLineHi and DataLineLo states.
* The StartRead signal must be active (StartRead = '1'), indicating the start of the SSI read cycle.

Note: The actual reading of valid data from the ssiDataOut line is dependent on these conditions and can be performed when all conditions are met simultaneously.

Here's an outline initialization procedure for the SSITop module:

1. **Setting Parameters 1**: **ParamWrite1** and **ParamWrite2** should be set to send a 60 MHz activation pulse. This will enable the writing of parameters into the **intDATA** signal on a rising edge of the **H1\_CLKWR** clock signal. **TransducerSelect** is set to **SSIBinaryAnalog**, which equals 6 (**110**). **DataLength** is set to 8.
2. **Setting Parameters 2**: **ParamWrite2** should be set to send a second 60 MHz activation pulse, which will be slightly offset from the first pulse, say one 60 MHz clock period. This will enable the writing of the next set of parameters. **DelayTerminalCount** is set to 0, and **HalfPeriod** is set to 2.
3. **Beginning SSI Read Cycle**: The **StartRead** signal can be asserted right away because **DelayTerminalCount** is set to 0.
4. **Reading Data**: During the SSI read cycle, the **SSI\_DATA** signal is sampled at the falling edge of the internally generated SSI clock (**intSSI\_CLK**) if **Shift** signal is active. This process continues until the **ShiftCounter** matches **DataLength**, indicating the required number of bits that have been read.
5. **SynchedTick Pulses**: **SynchedTick** will need to send two 30 MHz pulses. The first pulse is sent at 2 μs and the second pulse at 10 μs. This process leads to the latching of **Serial2ParallelData** contents into **SSIDataLatch**, and **Serial2ParallelData** is cleared.
6. **Ending the Read Cycle**: The read cycle is terminated when **ShiftCounter** matches **DataLength**, and the internal data valid (**intDataValid**) status is cleared.
7. **Data Output**: After the read cycle, if **SynchedTick** signal is asserted, the contents of the **Serial2ParallelData** register are latched into **SSIDataLatch** and **Serial2ParallelData** is cleared.

Also ensure that:

* DataLength is set to 8.
* Clockrate is set to 0.
* DelayCounter is set to 0.
* HalfPeriod is set to 2.
* All of this should happen internally, these values should not be forced.

Correct initialization of the system will be visible through the functioning of the SSITop module. Here are some indicators that can confirm the correct initialization of the system:

1. \*\*SSI\_DATA Stream\*\*: After setting all the parameters and initiating the read cycle, we should see meaningful data being read from `Serial2ParallelData` into `SSI\_DATA` at the falling edge of the internally generated SSI clock (`intSSI\_CLK`). The data should be in the format we expect based on the `TransducerSelect` and `Datalength` parameters.

2. \*\*SSI Clock\*\*: The SSI clock (`SSI\_CLK`) should generate nine negative pulses as mentioned in the notes. The frequency of the SSI clock is defined by `HalfPeriod` parameter and it should be as per the set configuration.

3. \*\*Data Output\*\*: The data out signal (`ssiDataOut`) should hold the read data when `PositionRead` is '1' and SSI Select (`SSISelect`) is '1'. This implies that the data from the SSI transducer is being correctly latched into `SSIDataLatch` and is ready to be output.

4. \*\*Status\*\*: When `StatusRead` and `SSISelect` are both '1', the `ssiDataOut` port should show the status information. The status information should reflect the configuration we have set (ClockRate, DataLength, TransducerSelect, etc) as well as the `NoXducer` and `DataValid` flags.

5. \*\*DataValid\*\*: The `DataValid` signal should be set to '1' after successful data read and remain so until the `PositionRead` signal falls. This indicates that the data read cycle was successful and that the read data is valid.

6. \*\*SynchedTick pulses\*\*: The `SynchedTick` should generate two pulses at 30 MHz as per the provided intervals. The first pulse should occur at 2 microseconds and the second pulse should occur at 10 microseconds. These pulses are essential for controlling the timing of various operations.

7. \*\*Wire Break Detection\*\*: The Wire Break detection functionality should function correctly. If the `SSI\_DATA` line is not high at the start of the read cycle or not low at the end of the read cycle, a wire break should be detected and the corresponding signals `DataLineHi` and `DataLineLo` should reflect this.

**SSITop** interfaces with a Synchronous Serial Interface (SSI) transducer. Here's a general idea of what some signals should output based on the code:

1. **ssiDataOut**: This signal will depend on whether **PositionRead** or **StatusRead** is active, and if **SSISelect** is high.
   * If **PositionRead** and **SSISelect** are high, the value on **ssiDataOut** will be equal to the value on **SSIDataLatch**.
   * If **StatusRead** and **SSISelect** are high, **ssiDataOut** will be a 32-bit value made up of several fields (including **NoXducer**, **DataValid**, **ClockRate**, **Datalength**, **TransducerSelect**).
   * If neither **PositionRead** nor **StatusRead** are high (or **SSISelect** is low), **ssiDataOut** will be **X"0000\_0000"**.
2. **SSI\_CLK**: This is the clock signal for the SSI data transfer. It gets toggled (flipped from '0' to '1' or '1' to '0') every time **ToggleEn** and **ClkOn** are both high, otherwise it remains at its last value. It's also set to high during the **StartRead** process.
3. **SSISelect**: This will be high when the transducer selected is either SSIBinaryAnalog or SSIGrayAnalog, otherwise it will be low.
4. **NoXducer**: It will be high if there is a break in the wire or connection to the transducer (**DataLineHi** or **DataLineLo** is high), otherwise it will be low.

Remember, these values are all subject to change based on the input signals and the internal processes that run when the **H1\_CLKWR** or **SysClk** signals have a rising edge (transition from low to high).

The **SSIDataLatch** signal, as used in your VHDL code, is an internal register that holds the value of the data received from the SSI interface. Its value changes during the process driven by the **SysClk** signal, specifically when the **SynchedTick** signal is high. At this moment, the **Serial2ParallelData** signal is latched into **SSIDataLatch** and **Serial2ParallelData** is reset to 0.

In another part of the same process, **Serial2ParallelData** is updated (shifted) when the **Shift** signal is high, meaning that new data is being received from the SSI interface (the **SSI\_DATA** signal). The actual value of **SSIDataLatch** depends on the input **SSI\_DATA** (converted from serial to parallel and possibly from Gray to Binary) at the moments when **SynchedTick** is high.

As for the **ssiDataOut** signal, it depends on the state of **SSIDataLatch**, **PositionRead**, **SSISelect**, **NoXducer**, **DataValid**, **ClockRate**, **Datalength**, **TransducerSelect** and **StatusRead** signals.

The input signals that would be crucial to provide a concrete value for **SSIDataLatch** and thus **ssiDataOut** would include:

* **SSI\_DATA**: the raw SSI data input
* **PositionRead**: the read request for the position
* **StatusRead**: the read request for the status
* **SynchedTick**: the signal indicating when data latching and shifting should occur
* **SysClk**: the clock signal driving the main logic of the SSI interface
* **H1\_CLKWR**: the clock signal driving the parameter writing process
* **ParamWrite1**: the signal indicating the first parameter write operation
* **ParamWrite2**: the signal indicating the second parameter write operation
* **intData**: the input data for parameter writing
* **Enable**: the enabling signal for the SSI interface
* **SlowEnable**: the slower enabling signal
* **TransducerSelect**: the selection of the transducer (analog binary or Gray)

These signals control the behavior of the SSI interface and determine the value of the **SSIDataLatch** and thus the **ssiDataOut** signals. The actual values would depend on the specific state of the SSI interface at a given moment, and the sequence of input data and control signals over time.

Primarily interested in the ssiDataOut signal, which is dependent on a few other signals such as PositionRead, SSISelect, StatusRead, and some internal latches. Here are the expected values under different conditions:

1. Most importantly SSI\_CLK must be active and oscillating.
2. When PositionRead is '1' and SSISelect is '1': ssiDataOut will be the same as SSIDataLatch, i.e., the value stored in SSIDataLatch.
3. When StatusRead is '1' and SSISelect is '1': ssiDataOut will be a 32-bit word composed of multiple bits and fields, including NoXducer, DataValid, ClockRate, Datalength, and TransducerSelect signals.
4. Otherwise, ssiDataOut will be "0000\_0000".

When StatusRead is '1' and SSISelect is '1', the ssiDataOut is set to an elaborate 32-bit word composed of several fields. The values in these fields are determined by the current state of signals like NoXducer, DataValid, ClockRate, Datalength, and TransducerSelect.

The 'Serial2ParallelData' value is typically the result of a shift register operation in which incoming serial data (from an SPI master device for example) is converted into a parallel format. It acts as a temporary buffer where each new serial bit shifts the previously received bits along and eventually forms a full word (e.g. 8 or 16 bits, depending on the system's configuration).

ssiDataOut (output): This is the output of the module, which outputs either the SSIDataLatch contents or status information.

The 32-bit vector ssiDataOut is used to output either status information or data from the latch, depending on the values of PositionRead and SSISelect. When PositionRead and SSISelect are high, it outputs the content of SSIDataLatch. When StatusRead and SSISelect are high, it outputs status information (including whether there is no transducer connected (NoXducer), whether the data is valid (DataValid), the clock rate, data length, and transducer select settings). If neither PositionRead and SSISelect nor StatusRead and SSISelect are high, it outputs an all-zero vector.

The bits in ssiDataOut represent different pieces of information depending on the state:

When outputting latched data (PositionRead and SSISelect are high), all 32 bits contain data from the SSI transducer.

When outputting status information (StatusRead and SSISelect are high), the data is formatted as follows:

* 31 downto 20: "0" (12 bits)
* 19: NoXducer
* 18: DataValid
* 17 downto 16: "00" (2 bits)
* 15 downto 14: ClockRate
* 13 downto 8: DataLength “001000”
* 7 downto 5: "000" (3 bits)
* 4 downto 0: TransducerSelect “00110”

The given code represents an architecture of an entity named "SSITop" in VHDL. Here are some findings based on the code:

1. Port Declarations: The entity "SSITop" has several input and output ports, including clock signals, control signals, data signals, and SSI (Synchronous Serial Interface) signals.
2. Constants:
   * The code defines two constants for binary and gray analog values used in the SSI interface.
   * These constants are then converted to std\_logic\_vector types.
3. Signals: The code declares various signals used within the architecture. Some notable signals are:
   * SSIDataLatch: A 32-bit signal used to latch the SSI data.
   * CheckDataLo and CheckDataHi: Signals used to check and latch the data line status before and after data transfer.
   * Shift: A signal used to control the shifting of data into the shift register.
   * StartRead: A signal used to initiate the SSI read cycle.
   * TransducerSelect, DataLength, ClockRate, DelayTerminalCount, HalfPeriod: Signals used for local parameter storage.
   * SSIBinaryAnalog and SSIGrayAnalog: Signals representing the selected mode (binary or gray) in the SSI interface.
   * NoXducer: A signal indicating whether there is an active transducer or not.
   * SequenceOn, ClkOn, ShiftOn, ToggleEn, CheckDataDelay, intSSI\_CLK, LineBreakDelay: Signals used in the SSI controller logic.
4. Processes:
   * The architecture contains two processes triggered by different clock signals: H1\_CLKWR and SysClk.
   * The H1\_CLKWR process handles parameter writes, clearing of internal data valid status, and transferring data valid status to CPU-accessible registers.
   * The SysClk process handles various operations related to SSI read cycles, shifting of data, clock generation, wire break detection, and control signal updates.
5. SSI Data Output:
   * The architecture assigns values to the ssiDataOut signal based on the conditions specified in the code. The output is determined by the values in SSIDataLatch and other control signals (PositionRead, SSISelect, StatusRead).
6. SSI Clock and Data Signals:
   * The internal signal intSSI\_CLK is assigned to the SSI\_CLK output.
   * The SSI\_DATA input is used in the shifting of data into the shift register.
7. Control and Status Signals:
   * Various control and status signals are updated based on different conditions and clock edges.
   * These signals include StartRead, SequenceOn, ClkOn, ToggleEn, ShiftOn, CheckDataHi, CheckDataLo, and LineBreakDelay.
8. Signal Dependencies:
   * The behavior of some signals depends on the values of other signals. For example, StartRead is dependent on DelayCounter, SequenceOn depends on CheckDataLo, and so on.
9. Signal Assignments:
   * The architecture includes several signal assignments using conditional statements and logical operations.
   * These assignments control the behavior of signals based on different conditions and events.

Overall, the architecture represents a system that interfaces with an SSI module, performs data transfer, controls clock signals, latches data, and handles various control and status operations.

* For every synched tick pulse after the first, every pulse sent by synched tick should trigger nine negative pulses on SSI\_CLK.
* We need to write an 8-bit value to the SSI\_DATA line ‘01011010’.
* We need to write a separate process / procedure block that synchronizes SSI\_DATA to SSI\_CLK.
* After approximately 1 ns after the first rising edge, we can write our first bit.
* The SSI\_DATA signal should be synced with the negative pulses that appear on the SSI\_CLK, so that during each negative pulse, one bit of the 8-bit value is written into SSI\_DATA.
* During the last negative pulse on SSI\_CLK, a zero should be written to SSI\_DATA (probably an error or status bit)