*Project: RMC75E FPGA TEST BENCH*

*Module: Quad.vhd*

*Author: Satchel Hamilton*

*Company: Delta Motion*

*Date: 7/13/2023*

*Last updated: July 13, 2023*

**Quad Init Procedure:**

1. Delay on startup:
   * Wait for one 60MHz **SysClk** cycle.
2. Send first sync tick pulse:
   * Set the **SynchedTick** signal to '1' to generate a synchronization tick pulse.
   * Wait for one **SysClk** cycle.
   * Set the **SynchedTick** signal back to '0' to end the synchronization tick pulse.
3. Wait for one clock cycle:
   * Pause the stimulus process for a duration of 20 microseconds.
4. Send second sync tick:
   * Set the **SynchedTick** signal to '1' to generate another synchronization tick pulse.
   * Wait for one **SysClk** cycle.
   * Set the **SynchedTick** signal back to '0' to end the synchronization tick pulse.
5. Perform actions related to Exp0 functionality:
   * Wait for 5 microseconds.
   * Set **Exp0Quad\_Reg** signal to '1'.
   * Set bit 2 of the **intDATA** signal to '1'.
   * Set bits 9 to 7 of the **intDATA** signal to "010".
   * Set bits 13 to 11 of the **intDATA** signal to "010".
   * Set **Exp0QuadLEDStatusWrite** signal to '1'.
   * Wait for 5 microseconds.
   * Set **Exp0Quad\_Reg** signal to '0'.
   * Set **Exp0QuadLEDStatusWrite** signal to '0'.
6. Wait for 20 microseconds.
7. Send third sync tick:
   * Set the **SynchedTick** signal to '1' to generate the third synchronization tick pulse.
   * Wait for one **SysClk** cycle.
   * Set the **SynchedTick** signal back to '0' to end the synchronization tick pulse.
8. Perform actions related to Exp0 functionality:
   * Set **Exp0QuadCountRead** signal to '1'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadCountRead** signal to '0'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadInputRead** signal to '1'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadInputRead** signal to '0'.
   * Set **Exp0QuadHomeRead** signal to '1'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadHomeRead** signal to '0'.
   * Set **Exp0QuadLEDStatusRead** signal to '1'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadLEDStatusRead** signal to '0'.
   * Set **Exp0QuadLatch0Read** signal to '1'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadLatch0Read** signal to '0'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadLatch1Read** signal to '1'.
   * Wait for 5 microseconds.
   * Set **Exp0QuadLatch1Read** signal to '0'.

To configure the counter in the QuadXface module, you need to modify the following signals and processes in the architecture:

1. Signal **QuadCount**: This is a 16-bit signal that represents the count value. You can set or modify this signal to initialize the counter to a specific value or manipulate the count value during operation.
2. Process **SysClk**: This process is responsible for incrementing or decrementing the **QuadCount** based on the **Increment** and **Decrement** signals. You can add additional logic here to customize the counter behavior.
3. Signal **Increment** and **Decrement**: These signals determine whether the counter should be incremented or decremented based on the input signals A and B. You can modify the logic in the corresponding processes to change the behavior of the counter.
4. Signal **QuadDataOut**: This is the output signal that represents the current count value. You can modify the assignment to this signal to control the value that is outputted.

The data displayed on QuadDataOut depends on the values of the following signals:

CountRead: When CountRead is asserted ('1'), the QuadDataOut displays the value of QuadCount. The logic for this condition is specified as:

QuadDataOut(31 downto 0) <=

QuadSignExt(31 downto 16) & QuadLatch(15 downto 0) when CountRead = '1'

LEDStatusRead: When LEDStatusRead is asserted ('1'), the QuadDataOut displays various status bits. The logic for this condition is specified as:

QuadDataOut(31 downto 0) <=

"000000" & IllegalTransition & ZBreak & BBreak & ABreak & AccumOverflow &

'0' & Latch1Lat & Latch0Lat & HomeLat & DirectionLat & EdgeMode & LearnModeEnable &

Latch1ArmedState(2 downto 0) & Latch1InSel & Latch0ArmedState(2 downto 0) &

Latch0InSel & HomeTriggerType(2 downto 0) & HomeArm & IndexPolarity & HomePolarity

InputRead: When InputRead is asserted ('1'), the QuadDataOut displays the value of intDATA. The logic for this condition is specified as:

QuadDataOut(31 downto 0) <= intDATA

HomeRead: When HomeRead is asserted ('1'), the QuadDataOut displays the value of HomeReg. The logic for this condition is specified as:

QuadDataOut(31 downto 0) <= HomeSignExt(31 downto 16) & HomeReg(15 downto 0) when HomeRead = '1'

Latch0Read: When Latch0Read is asserted ('1'), the QuadDataOut displays the value of Latch0Reg. The logic for this condition is specified as:

QuadDataOut(31 downto 0) <= Latch0SignExt(31 downto 16) & Latch0Reg(15 downto 0) when Latch0Read = '1'

Latch1Read: When Latch1Read is asserted ('1'), the QuadDataOut displays the value of Latch1Reg. The logic for this condition is specified as:

QuadDataOut(31 downto 0) <= Latch1SignExt(31 downto 16) & Latch1Reg(15 downto 0) when Latch1Read = '1'

Based on the above analysis, the conditions required for correct data to be displayed on QuadDataOut are as follows:

* For valid quad count data:
* CountRead must be asserted ('1').
* For displaying LED status bits:
* LEDStatusRead must be asserted ('1').
* For displaying intDATA input:
* InputRead must be asserted ('1').
* For displaying HomeReg value:
* HomeRead must be asserted ('1').
* For displaying Latch0Reg value:
* Latch0Read must be asserted ('1').
* For displaying Latch1Reg value:
* Latch1Read must be asserted ('1').

Here's a breakdown of the input and output ports of the "Quad" module:

Inputs:

* H1\_CLKWR: 60MHz read/write clock signal.
* SysClk: 30MHz system clock signal.
* SynchedTick: Synchronized tick signal.
* intDATA: 32-bit input data signal.
* Various control signals for each quadrature interface and axis:
  + Exp0QuadCountRead, Exp1QuadCountRead, Exp2QuadCountRead, Exp3QuadCountRead: Signals to initiate the count read operation for each respective quadrature interface.
  + Exp0QuadLEDStatusRead, Exp1QuadLEDStatusRead, Exp2QuadLEDStatusRead, Exp3QuadLEDStatusRead: Signals to read the LED status for each respective quadrature interface.
  + Exp0QuadLEDStatusWrite, Exp1QuadLEDStatusWrite, Exp2QuadLEDStatusWrite, Exp3QuadLEDStatusWrite: Signals to write the LED status for each respective quadrature interface.
  + Exp0QuadInputRead, Exp1QuadInputRead, Exp2QuadInputRead, Exp3QuadInputRead: Signals to read the input for each respective quadrature interface.
  + Exp0QuadHomeRead, Exp1QuadHomeRead, Exp2QuadHomeRead, Exp3QuadHomeRead: Signals to read the home position for each respective quadrature interface.
  + Exp0QuadLatch0Read, Exp0QuadLatch1Read, Exp1QuadLatch0Read, Exp1QuadLatch1Read, Exp2QuadLatch0Read, Exp2QuadLatch1Read, Exp3QuadLatch0Read, Exp3QuadLatch1Read: Signals to read the latch values for each respective quadrature interface.
  + QA0CountRead, QA1CountRead: Signals to initiate the count read operation for each respective axis.
  + QA0LEDStatusRead, QA1LEDStatusRead: Signals to read the LED status for each respective axis.
  + QA0LEDStatusWrite, QA1LEDStatusWrite: Signals to write the LED status for each respective axis.
  + QA0InputRead, QA1InputRead: Signals to read the input for each respective axis.
  + QA0HomeRead, QA1HomeRead: Signals to read the home position for each respective axis.
  + QA0Latch0Read, QA0Latch1Read, QA1Latch0Read, QA1Latch1Read: Signals to read the latch values for each respective axis.
  + Various control and status signals for each axis:
    - QA0\_SigA, QA0\_SigB, QA0\_SigZ, QA0\_Home, QA0\_RegX\_PosLmt, QA1\_SigA, QA1\_SigB, QA1\_SigZ, QA1\_Home, QA1\_RegX\_PosLmt, QA1\_RegY\_NegLmt.
  + QA0AxisFault, QA1AxisFault: 3-bit signals representing the fault status for each respective axis.

Outputs:

* Exp0QuadDataOut, Exp1QuadDataOut, Exp2QuadDataOut, Exp3QuadDataOut: 32-bit output data signals for each respective quadrature interface.
* QuadA0DataOut, QuadA1DataOut: 32-bit output data signals for each respective axis.

Internal Signals:

* Exp0\_LineFault, Exp1\_LineFault, Exp2\_LineFault, Exp3\_LineFault: 3-bit internal signals representing the line fault status for each respective quadrature interface.

The "Quad" module instantiates multiple instances of the "QuadXface" component, connecting their ports to the corresponding signals from the "Quad" entity. The "QuadXface" component is responsible for processing quadrature signals and generating output data for each quadrature interface and axis.

QuadDataOut signal is a 32-bit output vector representing various status information and control settings related to the motion control operations. Here is a breakdown of the bits in the QuadDataOut signal:

QuadDataOut(31 downto 16): These bits represent the latched quadrature count value. They store the 16-bit count value that keeps track of the position or displacement based on the quadrature encoder input.

QuadDataOut(15 downto 7): These bits are not explicitly defined in the code snippet provided. It's likely that they are reserved for other control or status information, specific to the implementation of the QuadXface module.

QuadDataOut(6 downto 0): These bits represent various control settings and status information related to the motion control operations. The specific meaning of each bit may vary depending on the implementation details of the QuadXface module.

The 16th bit of QuadDataOut is assigned the value of RegistrationY. The line of code responsible for this assignment is:

RegistrationY is an input signal to the QuadXface module, and its value is determined by the external logic or other parts of the system.

----------------------------------------------------------------------------------------------------------------------------------------

* Note that SysClk is likely meant to be a 60Mhz signal, H1\_CLKWR is a 30Mhz signal, and SynchedTick is meant to synchronize the data latches. It needs to send one 30 Mhz pulse at 1 us and a second pulse at later time (8-30 us)
* Inputs: Each entity has multiple inputs, such as clocks (H1\_CLKWR and SysClk), SynchedTick, internal data (intDATA), various Read signals, Home and Registration signals, and signals A, B, Index, as well as several fault flags. These inputs should be properly defined and initialized in the test bench before the simulation starts.
* Faults: Each entity monitors line faults with signals Exp\*\_Quad\_FaultA and Exp\*\_Quad\_FaultB. The polarity is reversed for the line faults, so a '0' indicates a fault. Any fault should ideally be corrected before the simulation.
* Component Mapping: The entity Quad is primarily composed of four instances of the component QuadXface. Each QuadXface entity requires the proper mapping of input signals, many of which are derived from the Quad entity's input signals. The output of each QuadXface entity is mapped to a corresponding QuadDataOut signal.
* The SynchedTick signal is a common input to all components and it must be properly generated in the test bench for the components to function correctly.
* Quad Data Output: Each QuadXface entity produces an output signal, QuadDataOut, which is passed back to the Quad entity and assigned to the corresponding Exp\*QuadDataOut signal. These signals are expected to be monitored in the test bench to verify correct operation.

1. Lastly, the test bench should appropriately model the system's behaviour, including generating valid signal transitions, and supplying valid data and control signal sequences. It should also check for correct responses and handle exceptional conditions such as faults.

**Quadxface notes:**

The code provided describes an entity named **QuadXface** and an associated architecture **QuadXface\_arch**. The entity describes the inputs and outputs to the module while the architecture outlines how these inputs and outputs interact.

**Entity**

The entity **QuadXface** defines the interface of the module. Here is a description of each port:

* **H1\_CLKWR**: Clock signal.
* **SysClk**: System clock signal.
* **SynchedTick**: Synchronized tick signal, typically used for timed actions.
* **intDATA**: Input data as a 32-bit vector.
* **QuadDataOut**: Output data as a 32-bit vector.
* **CountRead**, **LEDStatusRead**, **LEDStatusWrite**, **InputRead**, **HomeRead**, **Latch0Read**, **Latch1Read**: Control signals used to select certain operations within the architecture.
* **Home**, **RegistrationX**, **RegistrationY**, **LineFault**, **A**, **B**, **Index**: Additional inputs to the module.

**Architecture**

The architecture **QuadXface\_arch** defines the logic of the module. There are numerous signals and constants declared, including **PosDir** and **NegDir** which are constants set to **'1'** and **'0'** respectively, and are likely to be used to denote positive and negative directions.

Multiple 16-bit signals are declared to hold values for latches and registers (**Latch0Reg**, **Latch1Reg**, **HomeReg**, **QuadLatch**, **QuadCount**). Additional 16-bit high signals are sign extended to 32-bit signals (**QuadSignExt**, **HomeSignExt**, **Latch0SignExt**, **Latch1SignExt**).

Various 1-bit std\_logic signals are declared, most of them possibly used as flags or control signals for various operations within the module. For example, **Increment** and **Decrement** might be used to control up/down counting, and **IllegalTransition** may be used to indicate an invalid state change.

The architecture also defines an array of signals like **QA**, **QB**, **QZ**, **QH**, **QL0**, **QL1**, that are each 4-bit and 3-bit std\_logic vectors respectively. They could be used to store intermediate states for processing.

Several assignments are made within the architecture. For instance, the high 16 bits of the 32-bit **QuadSignExt**, **HomeSignExt**, **Latch0SignExt**, **Latch1SignExt** signals are assigned from the 16th bit of their corresponding 16-bit signals (**QuadLatch**, **HomeReg**, **Latch0Reg**, **Latch1Reg**). These operations perform sign extension to allow 16-bit values to be represented in a 32-bit vector.

The **QuadDataOut** output is conditional on several signals such as **CountRead**, **LEDStatusRead**, **InputRead**, **HomeRead**, **Latch0Read**, **Latch1Read**. The multiplexer-like structure implemented through the **when**-**else** keywords allows different sources of data to be selected for output based on the condition being tested.

The first process in the architecture is clocked on the **H1\_CLKWR** signal and seems to be used to configure several aspects of the module's operation based on the **LEDStatusWrite** signal and **intDATA** input vector.  
  
**QuadCount**: This process increments or decrements the counts based on the A/B sequence. The logic is as follows:

* + If **SynchedTick** is True, QuadCount is reset to 0.
  + If **LatchedInc** is True, QuadCount is incremented by 1.
  + If **LatchedDec** is True, QuadCount is decremented by 1.
  + If none of the conditions are True, **PostCount** is set to 0.

1. **QuadLatch**: This process transfers the counts to the latch when the control loop ticks comes by.
2. **MaxPosNum** and **MaxNegNum** signals check for overflow of the transition counter. They are set to '1' when QuadCount equals X"7FFF" and X"8000", respectively.
3. **intAccumOverflow** logic handles overflow detection, and resets if SynchedTick is True.
4. **Direction** tracking is performed. The direction is '1' for positive direction (PosDir) and '0' for negative direction (NegDir) and is updated on the falling edge of the SysClk if there is an increment or decrement.
5. **HomeRisingArmed**, **HomeFallingArmed**, **HomeIndexArmed**, **HomeIndexHomeArmed**, and **HomeIndexNotHomeArmed** are set based on the **HomeTriggerType** and **HomeArm**.
6. **RisingHome** and **FallingHome** signals capture rising and falling edge transitions for the home input.
7. **RisingHomeEvent** and **FallingHomeEvent** are set when the respective edge transitions occur.
8. **IndexEdgeEvent** captures edge transitions of the Index pulse based on the direction of the axis.
9. **EdgeDetectInput** is set when **IndexEdgeEvent** is '1' and any of **HomeIndexArmed**, **HomeIndexHomeArmed**, or **HomeIndexNotHomeArmed** is '1'.
10. **IndexEdgeDetected** latches the detection of the edge of the Index Pulse, if the index is used as part of the active homing routine.
11. **intEdgeMode** is updated based on the QB signal during the learning mode.
12. **EdgeMode** and **intLearnModeDone** are updated on the rising edge of H1\_CLKWR and SysClk respectively.
13. **IndexEvent**, **IndexHomeEvent**, and **IndexNotHomeEvent** signals determine edge events and the home event for the index.
14. **CaptureHomeCounts** is set when a RisingHomeEvent, FallingHomeEvent, IndexEvent, IndexHomeEvent, or IndexNotHomeEvent occurs and the respective event is armed.
15. **HomeReg** captures the current counts when an Event occurs. It also manages the status bits for Home latch.
16. **Latch0Input**, **RisingLatch0**, **FallingLatch0**, **RisingLatch0Event**, **FallingLatch0Event**, **CaptureLatch0Counts**, and **Latch0Reg** are similar to Home related signals but for Latch0.
17. **Latch1Input**, **RisingLatch1**, **FallingLatch1**, **RisingLatch1Event**, **FallingLatch1Event**, **CaptureLatch1Counts**, and **Latch1Reg** are similar to Home related signals but for Latch1.
18. **ABreak**, **BBreak**, **ZBreak**, **AccumOverflow**, **IllegalTransition** are updated during the SysClk and includes 3 for context:

**More Notes:**

The detailed notes are categorized into two groups for Quad and QuadXface respectively:

**Quad:**

**Inputs**: Clocks (H1\_CLKWR and SysClk), SynchedTick, intDATA, various Read signals, Home and Registration signals, A, B, Index, and several fault flags.

**Faults**: Monitored by signals Exp*\_Quad\_FaultA and Exp*\_Quad\_FaultB. A '0' indicates a fault.

**Component Mapping**: Composed of four instances of QuadXface. Proper mapping of input signals required. Output of each QuadXface mapped to a corresponding QuadDataOut signal.

**SynchedTick signal**: Common input to all components. Must be properly generated in the test bench.

**Quad Data Output**: Output signal of each QuadXface entity. Passed back to Quad entity and assigned to the corresponding Exp\*QuadDataOut signal.

**QuadXface:**

**Inputs**: H1\_CLKWR, SysClk, SynchedTick, intDATA, Control signals (CountRead, LEDStatusRead, LEDStatusWrite, InputRead, HomeRead, Latch0Read, Latch1Read), Home, RegistrationX, RegistrationY, LineFault, A, B, Index.

**Outputs**: QuadDataOut

**Internal Signals**: PosDir, NegDir, Latch0Reg, Latch1Reg, HomeReg, QuadLatch, QuadCount, QuadSignExt, HomeSignExt, Latch0SignExt, Latch1SignExt, QA, QB, QZ, QH, QL0, QL1.

**Processes**:

* QuadCount: Increments or decrements the counts based on the A/B sequence.
* QuadLatch: Transfers the counts to the latch when the control loop tick comes by.
* Overflow Checks: MaxPosNum and MaxNegNum check for overflow of the transition counter.
* intAccumOverflow: Handles overflow detection and resets if SynchedTick is True.
* Direction Tracking: The direction is '1' for positive and '0' for negative, updated on the falling edge of the SysClk.
* Home and Latch Processes: Track and handle edge transitions and latch counts for the Home, Latch0 and Latch1 inputs.
* Break and Overflow Updates: ABreak, BBreak, ZBreak, AccumOverflow, IllegalTransition are updated during the SysClk.

The test bench must also ensure that appropriate data sequences and control signals are generated, and that correct responses are verified.

Quad Additional NOTES:

Inputs:

* H1\_CLKWR: Clock signal.
* SysClk: System clock signal.
* SynchedTick: Synchronized tick signal.
* intDATA: 32-bit input data signal.
* Exp0QuadCountRead, Exp1QuadCountRead, Exp2QuadCountRead, Exp3QuadCountRead:
* Signals to initiate the count read operation for each respective quadrature interface.
* Exp0QuadLEDStatusRead, Exp1QuadLEDStatusRead, Exp2QuadLEDStatusRead, Exp3QuadLEDStatusRead:
* Signals to read the LED status for each respective quadrature interface.
* Exp0QuadLEDStatusWrite, Exp1QuadLEDStatusWrite, Exp2QuadLEDStatusWrite, Exp3QuadLEDStatusWrite:
* Signals to write the LED status for each respective quadrature interface.
* Exp0QuadInputRead, Exp1QuadInputRead, Exp2QuadInputRead, Exp3QuadInputRead:
* Signals to read the input for each respective quadrature interface.
* Exp0QuadHomeRead, Exp1QuadHomeRead, Exp2QuadHomeRead, Exp3QuadHomeRead:
* Signals to read the home position for each respective quadrature interface.
* Exp0QuadLatch0Read, Exp0QuadLatch1Read, Exp1QuadLatch0Read, Exp1QuadLatch1Read,
* Exp2QuadLatch0Read, Exp2QuadLatch1Read, Exp3QuadLatch0Read, Exp3QuadLatch1Read:
* Signals to read the latch values for each respective quadrature interface.
* Exp0Quad\_A, Exp0Quad\_B, Exp1Quad\_A, Exp1Quad\_B, Exp2Quad\_A, Exp2Quad\_B,
* Exp3Quad\_A, Exp3Quad\_B: Signals representing the quadrature input signals for each respective quadrature interface.
* Exp0Quad\_Reg, Exp1Quad\_Reg, Exp2Quad\_Reg, Exp3Quad\_Reg:
* Signals representing the quadrature home position for each respective quadrature interface.
* Exp0Quad\_FaultA, Exp0Quad\_FaultB, Exp1Quad\_FaultA, Exp1Quad\_FaultB, Exp2Quad\_FaultA, Exp2Quad\_FaultB,
* Exp3Quad\_FaultA, Exp3Quad\_FaultB: Signals representing the fault status for each respective quadrature interface.
* QA0CountRead, QA1CountRead: Signals to initiate the count read operation for each respective axis.
* QA0LEDStatusRead, QA1LEDStatusRead: Signals to read the LED status for each respective axis.
* QA0LEDStatusWrite, QA1LEDStatusWrite: Signals to write the LED status for each respective axis.
* QA0InputRead, QA1InputRead: Signals to read the input for each respective axis.
* QA0HomeRead, QA1HomeRead: Signals to read the home position for each respective axis.
* QA0Latch0Read, QA0Latch1Read, QA1Latch0Read, QA1Latch1Read:
* Signals to read the latch values for each respective axis.
* QA0\_SigA, QA0\_SigB, QA0\_SigZ, QA0\_Home, QA0\_RegX\_PosLmt, QA1\_SigA,
* QA1\_SigB, QA1\_SigZ, QA1\_Home, QA1\_RegX\_PosLmt, QA1\_RegY\_NegLmt:
* Signals representing various control and status signals for each respective axis.
* QA0AxisFault, QA1AxisFault:
* 3-bit signals representing the fault status for each respective axis.

Outputs:

* Exp0QuadDataOut, Exp1QuadDataOut, Exp2QuadDataOut, Exp3QuadDataOut: 32-bit output data signals for each respective quadrature interface.
* QuadA0DataOut, QuadA1DataOut: 32-bit output data signals for each respective axis.
* Internal Signals:
* Exp0\_LineFault, Exp1\_LineFault, Exp2\_LineFault, Exp3\_LineFault:
* 3-bit internal signals representing the line fault status for each respective quadrature interface.
* The architecture section of the module contains multiple instances of a component called "QuadXface."
* Each instance represents a quadrature interface or axis and is responsible
* for processing the respective signals and generating the output data.

The QuadXface component has the following ports:

* H1\_CLKWR: Clock signal.
* SysClk: System clock signal.
* SynchedTick: Synchronized tick signal.
* intDATA: 32-bit input data signal.
* QuadDataOut: 32-bit output data signal.
* CountRead: Signal to initiate the count read operation.
* LEDStatusRead: Signal to read the LED status.
* LEDStatusWrite: Signal to write the LED status.
* InputRead: Signal to read the input.
* HomeRead: Signal to read the home position.
* Latch0Read: Signal to read the first latch value.
* Latch1Read: Signal to read the second latch value.
* Home: Signal representing the home position.
* RegistrationX: Signal representing the X-axis registration.
* RegistrationY: Signal representing the Y-axis registration.
* LineFault: 3-bit signal representing the line fault status.
* A, B: Signals representing the quadrature input signals.
* Index: Signal representing the quadrature index signal.
* The QuadXface components are instantiated for each quadrature interface and axis,
* and their ports are connected to the corresponding signals from the Quad entity.
* The instantiation maps the signals to the appropriate inputs and outputs of the QuadXface components.
* Overall, the Quad module serves as a wrapper for multiple QuadXface components,
* enabling the processing of quadrature signals and generating output data for multiple
* quadrature interfaces and axes in the RMC75E motion controller.

The Quad module is a top-level entity that instantiates four instances of the QuadXface component.

It handles inputs, outputs, faults, and component mapping for the QuadXface instances.

Inputs to the Quad module include clocks (H1\_CLKWR and SysClk), synchronization tick (SynchedTick), data (intDATA), various control and read signals, and fault flags.

Fault flags (Exp\_Quad\_FaultA and Exp\_Quad\_FaultB) indicate the presence of faults.

The Quad module maps inputs and outputs between the QuadXface instances and assigns the output data signals (QuadDataOut) to corresponding signals (Exp\*QuadDataOut).

QuadXface Module:

The QuadXface module defines the interface and logic of a single quadrature interface or axis.

Inputs to the QuadXface module include clocks (H1\_CLKWR and SysClk), synchronization tick (SynchedTick), data (intDATA), control and read signals, and various input signals (Home, RegistrationX, RegistrationY, LineFault, A, B, Index).

The QuadXface module has an output signal called QuadDataOut, representing the output data.

The architecture of QuadXface includes multiple processes and internal signals for handling different operations and state management.

Some processes are responsible for counting, latch transfer, overflow detection, direction tracking, and edge event detection.

The architecture also includes various internal signals used for intermediate storage and processing of data.

The QuadDataOut signal is generated based on different conditions and control signals using multiplexers.

The architecture includes logic for handling faults, edge events, latch captures, and various control operations.

Test Bench:

The test bench for the Quad module should initialize and define all input signals.

It should monitor fault signals and ensure faults are corrected before the simulation.

Component mapping should be properly established in the test bench.

The SynchedTick signal should be generated correctly in the test bench.

The test bench should monitor the QuadDataOut signals to verify correct operation.

It should model the system behavior, generate valid signal transitions, supply valid data and control signal sequences, and handle exceptional conditions such as faults.

The main functionality of the QuadXface module can be summarized as follows:

1. Quadrature Encoding: The module processes the input signals A and B, which represent the quadrature encoder channels. It detects changes in the state of these signals and determines the direction of rotation (positive or negative).
2. Counting: The module maintains a 16-bit count value, referred to as QuadCount, which keeps track of the position or displacement based on the quadrature encoder input. It increments or decrements this count value based on the detected direction of rotation.
3. Status Management: The module handles various status indicators and flags related to the motion control operations. These include ZBreak, ABreak, BBreak, AccumOverflow, and IllegalTransition, which provide information on fault conditions, count overflow, and illegal state transitions.
4. Edge Detection: The module detects rising and falling edges of specific input signals, such as Home and Index. It generates corresponding edge events, such as RisingHomeEvent, FallingHomeEvent, IndexEvent, IndexHomeEvent, and IndexNotHomeEvent, based on the configured edge modes and homing arming conditions.
5. Latch Capture: The module captures the current count value (QuadCount) when specific latch triggers are activated. These triggers include RisingLatch0Event, FallingLatch0Event, and RisingLatch1Event, which capture the count value on rising or falling edges of latch inputs. The captured values are stored in Latch0Reg and Latch1Reg, respectively.
6. Home and Registration Control: The module supports homing operations by monitoring the Home input signal. It also handles registration inputs (RegistrationX and RegistrationY) for position reference purposes.
7. Configuration and Communication: The module allows for configuration and communication through input ports such as LEDStatusRead, LEDStatusWrite, InputRead, and intDATA. These inputs enable the configuration of parameters related to polarity, arming, edge modes, and other control settings.

Top of Form