# RMC75E Module Overview

---

| | |
|---|---|
| Entity Name | Top |
| File | top.vhd |

Description:

- This design provides the interface to all the axis modules and expansion
- modules on the RMC75E as well as control of the LEDs on the CPU module.
- The main components are:
- bus interface to the MPC5200 processor
- sensor interface logic
- sensor I/O signal multiplexing

---

---

Entity Name: SerialMemoryInterface

File: SerialMemoryInterface.vhd

Description: The SerialMemoryInterface module is a component of the RMC75E modular motion controller's source code. It provides an interface for serial communication with AT24C01A serial EEPROM modules. This module facilitates read and write operations to the EEPROM by managing address selection, data transfer, and clock generation. The core functionality is driven by a state machine that controls the serial communication protocol.

The module's input ports include SysReset, H1_CLK, SysClk, SlowEnable, intDATA, SerialMemXfaceWrite, SerialMemoryDataIn, and SerialMemoryDataControl. These signals are used to control the module's operation and transfer data. The output ports include serialMemDataOut, SerialMemoryDataOut, SerialMemoryClk, Exp0SerialSelect, Exp1SerialSelect, Exp2SerialSelect, Exp3SerialSelect, EEPROMAccessFlag, M_SPROM_CLK, and M_SPROM_DATA. These signals provide data output and control signals for external components.

The module utilizes various constants to configure its behavior, such as DeviceAddress, SerialMemoryClockTerminalCount, SerialMemoryClockTerminalCountDiv2, Exp0SelectAddr, Exp1SelectAddr, Exp2SelectAddr, Exp3SelectAddr, and ControlModuleSelectAddr. These constants define the device address, clock terminal count values, selection addresses, and control module select address.

The state machine within the module governs the serial communication protocol. It is composed of multiple states, including IdleState, SignalStartState, DeviceAddrState, CheckDeviceAddrACKState, MemAddrState, CheckMemoryAddrACKState, OperationTypeState, WriteState, WriteAckState, ReadState, ReadNoACKState, StopState, and ClearState. These states control the sequence of operations during read and write operations to the EEPROM.

Several internal signals are used to facilitate the module's functionality. These signals include LoadDeviceAddr, LoadMemAddr, LoadWriteData, ShiftDone, SecondPassRead, ReadLatch, WriteLatch, WriteFlag, FLAG_CLR, intFLAG_CLR, FLAG_CLR_LAT, ReadFlag, SerialDataCounter, SerialMemoryClockEnableCounter, SerialMemoryClockEnable, SerialDataInputEnable, intSerialMemoryDataIn, StartStopBit, ACK, MemoryAddress, intModuleAddress, DataBuffer, intSerialMemoryClock, ShiftEnable, intSerialMemoryDataControl, intSerialMemoryDataOut, SerialMemoryClockFallingEdge, SerialMemoryMidClockEnable, intEEPROMAccessFlag, ControlSerialSelect, SerialDataInput, and SerialDataOutput.

The module also includes supporting logic for the state machine, clock generation, flag management, data transfer, and fault handling. It utilizes counters, registers, and control signals to coordinate the read and write operations with the EEPROM.

Overall, the SerialMemoryInterface module provides a reliable and efficient interface for serial communication with AT24C01A serial EEPROM modules within the RMC75E modular motion controller. It handles the intricate details of address selection, data transfer, and clock generation, allowing seamless integration of EEPROM functionality into the larger motion control system.

---

Entity Name: TickSync

File: TickSync.vhd

Description:

The TickSync module is part of the source code for the RMC75E modular motion controller. Its purpose is to synchronize the incoming control loop clock tick with the internal system clock, making it synchronous and usable by the internal logic. It provides two synchronized output pulses, SynchedTick and SynchedTick60, which are aligned with the rising edge of the 30MHz and 60MHz system clocks, respectively.

The module takes several input signals, including SysReset, SysClk, H1_CLK, and LOOPTICK. SysReset is the main system reset signal, SysClk is the 30MHz system clock, H1_CLK is the 60MHz system clock, and LOOPTICK is the incoming control loop clock tick. The module outputs two synchronized pulses, SynchedTick and SynchedTick60, which represent the synchronized control loop timing pulses.

Internally, the module uses two signals, TickSync and LatchedTickSync, for synchronizing the control loop tick with the system clock. A process is triggered by changes in SysClk and LOOPTICK to update the TickSync signal. On the rising edge of SysClk, the LatchedTickSync signal is updated based on the current state of TickSync and SysReset. The SynchedTick output is then assigned the value of LatchedTickSync.

Similarly, a separate process is triggered by changes in H1_CLK and LOOPTICK to synchronize the control loop tick with the 60MHz system clock. The TickSync60 and LatchedTickSync60 signals are used for this purpose. On the rising edge of H1_CLK, the LatchedTickSync60 signal is updated based on the current state of TickSync60 and SysReset. The SynchedTick60 output is assigned the value of LatchedTickSync60.

The module effectively synchronizes the control loop timing pulse with the internal system clocks, allowing precise timing control for various internal logic operations. It ensures that events and operations within the modular motion controller are aligned with the system clocks, enhancing the overall functionality and performance of the controller.

---

-- Entity Name: Decode

-- File: Decode.vhd

Description:

The Decode module is responsible for generating the WRITE control lines and READ control lines for the RMC75E modular motion controller's source code. It decodes the address signals and other input signals to determine the appropriate control signals to enable or disable various functionalities and register accesses.

The module takes multiple input signals, including ADDR (address), RD_L (read control), WR_L (write control), CS_L (chip select), MDTPresent (MDT present signal), ANLGPresent (analog present signal), QUADPresent (quadrature present signal), Exp0Mux (expansion 0 multiplexer selection), Exp1Mux (expansion 1 multiplexer selection), Exp2Mux (expansion 2 multiplexer selection), and Exp3Mux (expansion 3 multiplexer selection).

The module provides a wide range of output signals, including FPGAAccess (FPGA access control signal), ANLGAccess (analog access control signal), FPGAIDRead (FPGA ID read control signal), CPUConfigRead (CPU configuration read control signal), CPUConfigWrite (CPU configuration write control signal), WDTConfigRead (watchdog timer configuration read control signal), WDTConfigWrite (watchdog timer configuration write control signal), CPULEDRead (CPU LED read control signal), CPULEDWrite (CPU LED write control signal), FPGAProgrammedRead (FPGA programmed read control signal), FPGAProgrammedWrite (FPGA programmed write control signal), ControlCardIDRead (control card ID read control signal), Expansion1IDRead (Expansion 1 ID read control signal), Expansion2IDRead (Expansion 2 ID read control signal), Expansion3IDRead (Expansion 3 ID read control signal), Expansion4IDRead (Expansion 4 ID read control signal), MDT_SSIPositionRead0 (MDT SSI position read control signal for channel 0), MDT_SSIStatusRead0 (MDT SSI status read control signal for channel 0), MDT_SSIConfigWrite0 (MDT SSI configuration write control signal for channel 0), and many more.

The module includes constants that represent the specific addresses for different registers and components in the motion controller. These addresses are used for decoding the input address signal to determine the corresponding control signals for each component. Some of the important constants include FPGAIDAddr, CPUConfigAddr, CPULEDAddr, WDTConfigAddr, PROFIBUSAddr, SerialMemXfaceAddr, ControlCardIDAddr, FPGAProgrammedAddr, Expansion1IDAddr, Expansion2IDAddr, Expansion3IDAddr, Expansion4IDAddr, LatencyCounterAddr, Axis0x010Addr, Axis0x011Addr, Axis0x012Addr, Axis0x013Addr, Axis0x014Addr, Axis0x015Addr, Axis0x016Addr, Axis1x020Addr, Axis1x021Addr, Axis1x022Addr, Axis1x023Addr, Axis1x024Addr, Axis1x025Addr, Axis1x026Addr, Expansion0Addr, Expansion1Addr, Expansion2Addr, Expansion3Addr, ExpD8ConfigAddr, ExpD8DinAddr, Exp0AnlgCh0Addr, Exp0AnlgLED0Addr, Exp0AnlgCh1Addr, Exp0AnlgLED1Addr, Exp1AnlgCh0Addr, Exp1AnlgLED0Addr, Exp1AnlgCh1Addr, Exp1AnlgLED1Addr, Exp2AnlgCh0Addr, Exp2AnlgLED0Addr, Exp2AnlgCh1Addr, Exp2AnlgLED1Addr, Exp3AnlgCh0Addr, Exp3AnlgLED0Addr, Exp3AnlgCh1Addr, Exp3AnlgLED1Addr, Exp0QuadCountAddr, Exp0QuadLEDStatusAddr, Exp0InputAddr, Exp0HomeAddr, Exp0Latch0Addr, Exp0Latch1Addr, Exp1QuadCountAddr, Exp1QuadLEDStatusAddr, Exp1InputAddr, Exp1HomeAddr, Exp1Latch0Addr, Exp1Latch1Addr, Exp2QuadCountAddr, Exp2QuadLEDStatusAddr, Exp2InputAddr, Exp2HomeAddr, Exp2Latch0Addr, Exp2Latch1Addr, Exp3QuadCountAddr, Exp3QuadLEDStatusAddr, Exp3InputAddr, Exp3HomeAddr, Exp3Latch0Addr, Exp3Latch1Addr.

The module includes signals that are used for generating control signals based on the decoded address and input signals. These signals include Axis0x010Read, Axis0x011Read, Axis0x012Read, Axis0x014Read, Axis0x015Read, Axis0x016Read, Axis0x010Write, Axis0x011Write, Axis0x012Write, Axis0x015Write, Axis1x020Read, Axis1x021Read, Axis1x022Read, Axis1x024Read, Axis1x025Read, Axis1x026Read, Axis1x020Write, Axis1x021Write, Axis1x022Write, Axis1x025Write, Exp0Read, Exp1Read, Exp2Read, Exp3Read, Exp0Write, Exp1Write, Exp2Write, Exp3Write, Exp0AnalogRead, Exp1AnalogRead, Exp2AnalogRead, Exp3AnalogRead, Exp0AnalogWrite, Exp1AnalogWrite, Exp2AnalogWrite, Exp3AnalogWrite, Exp0DIO8ConfigRead, Exp0DIO8ConfigWrite, Exp1DIO8ConfigRead, Exp1DIO8ConfigWrite, Exp2DIO8ConfigRead, Exp2DIO8ConfigWrite, Exp3DIO8ConfigRead, Exp3DIO8ConfigWrite, Exp0DIO8DinRead, Exp1DIO8DinRead, Exp2DIO8DinRead, Exp3DIO8DinRead, Exp0QuadRead, Exp0QuadWrite, Exp1QuadRead, Exp1QuadWrite, Exp2QuadRead, Exp2QuadWrite, Exp3QuadRead, Exp3QuadWrite, ExpA0LED0Read, ExpA0LED1Read, ExpA0LED0Write, ExpA0LED1Write, ExpA1LED0Read, ExpA1LED1Read, ExpA1LED0Write, ExpA1LED1Write, ExpA2LED0Read, ExpA2LED1Read, ExpA2LED0Write, ExpA2LED1Write, ExpA3LED0Read, ExpA3LED1Read, ExpA3LED0Write, ExpA3LED1Write.

The logic of the module is based on the input signals and the decoded address. It sets the appropriate control signals to '1' when the conditions for a specific component or register access are met. For example, FPGAAccess is set to '1' when CS_L is low and either RD_L or WR_L is low, indicating a memory access to the FPGA. ANLGAccess is set to '1' when any analog-related data access signal is active. The other control signals follow similar conditions based on the input signals and decoded address.

Overall, the Decode module enables the generation of control signals for different components and register accesses based on the input address and other signals

---

| | |
|---|---|
| Entity Name | CPUConfig |
| File | CPUConfig.vhd |

Description: The entity "CPUConfig" is a module in the source code for the RMC75E modular motion controller. It is implemented in VHDL and is responsible for handling CPU configuration and control signals.

The module has the following ports:

- RESET: An input signal representing the external reset from the power monitor and watchdog IC.

- SysRESET: An input signal representing the main internal reset used to clear DLL_RST.

- H1_CLKWR: An input signal related to clock synchronization.

- H1_PRIMARY: An input signal representing the 60MHz system clock.

- intDATA: An input signal representing internal data.

- cpuConfigDataOut: An output signal representing the CPU configuration data output.

- CPUConfigWrite: An input signal used for initiating CPU configuration writes.

- M_DRV_EN_L: An output signal representing the control output enable line.

- HALT_DRIVE_L: An input signal related to drive control.

- DLL_LOCK: An input signal indicating the status of the DLL (Delay-Locked Loop).

- DLL_RST: An output signal used to clear SysRESET and originally for resetting DLL.

- LoopTime: An output signal representing the control loop time selection.

- ENET_Build: An input signal related to Ethernet configuration.

The architecture "CPUConfig_arch" contains the implementation details of the module.

The module includes several internal signals such as int_M_DRV_EN, int_DLL_RST, intLoopTime, dll_rst_pre_queue, and dll_rst_queue.

The LoopTime signal is assigned the value of intLoopTime.

The cpuConfigDataOut signal is assigned a concatenation of different signals and bits from the internal signals. It represents the CPU configuration data output.

The process within the module handles various operations based on the RESET and H1_CLKWR signals. It controls the int_M_DRV_EN signal based on the HALT_DRIVE_L and CPUConfigWrite signals.

The intLoopTime signal is updated based on the intDATA input when CPUConfigWrite is high.

Another process handles the DLL_RST signal and ensures proper synchronization and glitch protection. It updates the int_DLL_RST signal based on SysRESET and CPUConfigWrite. The dll_rst_pre_queue and dll_rst_queue signals provide glitch protection, and the DLL_RST output signal is assigned the appropriate value from dll_rst_queue.

Overall, the CPUConfig module manages CPU configuration and control signals, including enabling or disabling drives, handling resets, and configuring loop time selection. It ensures proper synchronization and glitch protection during the configuration process.

---

Entity Name ControlOutput

**File ControlOutput.vhd**

Description:

The entity "ControlOutput" is a module in the source code for the RMC75E modular motion controller. It is implemented in VHDL and represents the analog control output interface.

The module has the following ports:

- H1_CLKWR: An input signal related to clock synchronization.
- SysClk: An input signal representing the system clock.
- RESET: An input signal representing the external reset from the power monitor and watchdog IC.

- SynchedTick: An input signal used for synchronization.
- intDATA: An input signal representing the control output data.
- ControlOutputWrite: An input signal used to initiate control output writes.
- M_OUT_CLK: An output signal representing the clock for the control output interface.
- M_OUT_DATA: An output signal representing the data for the control output interface.
- M_OUT_CONTROL: An output signal representing the control signal for the control output interface.
- PowerUp: An input signal related to power-up status.
- Enable: An input signal used to enable or disable the control output interface.

The architecture "ControlOutput_arch" contains the implementation details of the module.

The module includes several internal signals such as ShiftRegister, DataBuffer, DataBufferOut, Count, ControlOutputWriteLatched0, ControlOutputWriteLatched1, ControlOutputWriteLatched2, ControlOutputOneShot, OutputClock, ShiftEnable, ShiftComplete, and ShiftDataOutput.

The process within the module handles the DataBuffer and ControlOutputWrite signals. It loads the control output data from intDATA into the DataBuffer when ControlOutputWrite is high. The ControlOutputWriteLatched signals are used as edge detectors to detect the falling edge of ControlOutputWrite and trigger the shift process.

The DataBufferOut signal represents the converted control output data in MAX5541 format. It is derived from the DataBuffer and ensures proper formatting for the DAC.

Another process handles the SysClk signal and updates the ControlOutputWriteLatched signals accordingly. It also determines the ControlOutputOneShot signal based on the ControlOutputWriteLatched signals and PowerUp status.

The StateMachine process controls the write sequence to the DAC outputs. It utilizes a state variable "State" and transitions between the reset state (s0) and the shift state (s1). In the reset state, the data shifting is disabled, and the control output signals are set accordingly. In the shift state, the control output signals are updated based on the ShiftComplete and ShiftEnable signals. The StateMachine process ensures proper synchronization and control during the data shifting process.

The process related to the SysClk signal handles the output clock generation and the 5-bit synchronous counter. It determines the OutputClock signal based on SynchedTick and ShiftEnable. It updates the Count signal based on SynchedTick and OutputClock. When the count reaches the terminal count (16), the ShiftComplete signal is set. The ShiftRegister is loaded with the DataBufferOut during the ControlOutputOneShot condition, and it is shifted when ShiftEnable is high and OutputClock is low. This process controls the data shifting and ensures proper synchronization.

The M_OUT_CLK signal is assigned the OutputClock, representing the clock for the control output interface.

The ShiftDataOutput signal represents the MSB of the ShiftRegister and is assigned to M_OUT_DATA, representing the data for the control output interface.

The ControlOutput module manages the control output interface by loading control output data into a shift register and shifting it out serially to the DAC. It handles synchronization, edge detection, and state transitions to ensure proper control output operation.

---

Entity Name ExpSigRoute

File ExpansionSigRoute.vhd

Description:

The entity "ExpSigRoute" represents a module in the source code for the RMC75E modular motion controller. The module is implemented in the VHDL language and is responsible for routing various signals within the expansion module.

The module has the following ports:

- ExpMux: An input signal used for selecting the routing configuration.
- ExpSerialSelect: An input signal used for selecting the serial configuration.
- ExpLEDSelect: An input signal used for selecting the LED configuration.
- ExpLEDData: An input signal representing the data input for the LED configuration.
- ExpData: A bidirectional signal representing the routed signals.
- ExpA_CS_L: An input signal used for controlling the chip select in the analog input configuration.
- ExpA_CLK: An input signal representing the clock in the analog input configuration.
- ExpA_DATA: An output signal representing the data output in the analog input configuration.
- SerialMemoryDataIn: An output signal representing the data input for the serial memory configuration.
- SerialMemoryDataOut: An input signal representing the data output from the serial memory configuration.
- SerialMemoryDataControl: An input signal used for controlling the data in the serial memory configuration.
- SerialMemoryClk: An input signal representing the clock in the serial memory configuration.
- ExpD8_DataIn: An output signal representing the data input for the DIO (Digital Input/Output) configuration.
- ExpD8_Clk: An input signal representing the clock in the DIO configuration.
- ExpD8_DataOut: An input signal representing the data output from the DIO configuration.
- ExpD8_OE: An input signal used for enabling the output in the DIO configuration.
- ExpD8_Load: An input signal used for loading data in the DIO configuration.

- ExpD8_Latch: An input signal used for latching data in the DIO configuration.
- ExpQ1_A, ExpQ1_B, ExpQ1_Reg, ExpQ1_FaultA, ExpQ1_FaultB: Output signals representing different outputs in the Q1 logic module.

The architecture "ExpSigRoute_arch" of the module contains the implementation details.

The module includes several constants to define values used within the implementation, such as NA_Value, Ax_Value, D8_Value, and Q1_Value. These constants represent different configurations and are converted to std_logic_vector types.

The module defines the routing logic based on the ExpMux settings. The ExpData signal is assigned different values depending on the selected configuration. The ExpQ1_A, ExpQ1_B, ExpQ1_Reg, ExpQ1_FaultA, and ExpQ1_FaultB signals are assigned values from the ExpData signal when the ExpMux is set to Q1.

In the analog input configuration, the SerialMemoryDataIn signal is assigned the value from the ExpData signal when the ExpMux is set to Ax and ExpSerialSelect is '1'. Otherwise, it is assigned '0' or 'Z' based on the conditions.

The ExpA_DATA signal is assigned values from the ExpData signal when the ExpMux is set to Ax.

In the DIO input configuration, the ExpD8_DataIn signal is assigned the value from the ExpData signal when the ExpMux is set to D8.

The ExpData signal's different bits (0 to 5) are assigned values based on the selected configuration. This includes routing the signals from the SerialMemoryClk, ExpD8_Clk, ExpLEDData, ExpD8_DataOut, ExpD8_Latch, ExpD8_OE, ExpA_CS_L, ExpA_CLK, ExpD8_Load, and ExpLEDSelect signals to the appropriate bits of the ExpData signal.

Overall, the ExpSigRoute module provides a flexible signal routing mechanism for different configurations within the expansion module. It allows for selecting the appropriate signals based on the configuration settings and ensures proper signal routing and selection based on the chosen configuration.

---

Entity Name: DiscoverControlID

File: DiscoverControlID.vhd

Description:

The entity "DiscoverControlID" represents a module in the source code for the RMC75E modular motion controller. The module is implemented in the VHDL language and is responsible for controlling the discovery of the control ID for the motion controller.

The module has the following ports:

- RESET: An input signal used to reset the module.
- SysClk: An input signal representing the system clock.

- SlowEnable: An input signal that enables the slow clock output for serial communication.
- ControlID: A bidirectional signal that represents the control ID of the motion controller.
- M_Card_ID_CLK: An output signal used for clocking the M_Card_ID_DATA signal.
- M_Card_ID_DATA: An input signal representing the data input for discovering the control ID.
- M_Card_ID_LATCH: An output signal used for latching the M_Card_ID_DATA signal.
- M_Card_ID_LOAD: An output signal used for loading the M_Card_ID_DATA signal.

The architecture "DiscoverControlID_arch" of the module contains the implementation details.

The module uses several constants to define values used within the implementation. These include "TerminalCountValue" and "TerminalCount" constants used for terminal count comparison, and "s0_LatchState" to "s5_StopState" constants representing different states of the state machine.

The module includes several signals:

- State: A signal representing the current state of the state machine.
- Count: A signal representing a 4-bit synchronous counter.
- OutputClock: A signal used as the clock for shifting the control ID.
- ShiftEnable: A signal used to enable the shifting of the control ID.
- ShiftComplete: A signal indicating the completion of the shifting process.

The module contains a process called "StateMachine" that controls the state transitions based on the system clock. It handles the sequencing of operations required for discovering the control ID. The process includes different cases for each state, setting appropriate signals based on the current state.

Another process in the module, called "process", handles the shifting of the control ID bits and the counting operation. It updates the ControlID signal based on the input clock signals and the current state. It also checks for the terminal count and sets the ShiftComplete signal accordingly.

The module assigns the OutputClock signal to the M_Card_ID_CLK signal, representing the clock for the M_Card_ID_DATA input.

The ControlID signal's 16th bit is used as a data valid flag, which is set to '1' when the state is in the s5_StopState.

Overall, the DiscoverControlID module controls the discovery of the control ID for the motion controller through a state machine and shifting operations. It ensures the proper sequencing of operations and provides the necessary control signals for discovering the control ID.

---

Entity Name: CPULED

File: CPULED.vhd

Description: -- The CPULED module is part of the source code for the RMC75E modular motion controller. It provides a simple implementation for driving the CPU LED indicators. The module receives input data from the FPGA and latches the bits to drive the LEDs. The CPULEDWrite signal enables writing to the LED data. If modulation is required, it can also be implemented within this module.

---

Entity Name: MDTSSIRoute

File: mdtssiroute.vhd

Description: MDT/SSI Routing

The MDTSSIRoute module describes the interface and behavior of a routing module for the Magnetostrictive Displacement Transducer (MDT) and Synchronous Serial Interface (SSI) clocks in the RMC75E modular motion controller. The module takes input signals related to the SSI select, internal MDT clock, SSI clock, and MDT interrupt, and provides output signals for the internal clocks of Axis0 and Axis1.

The purpose of the MDTSSIRoute module is to control the routing of clock signals for the MDT and SSI based on the SSI select signal. It selects either the MDT internal clock or the SSI clock for each axis based on the value of the SSI select signal.

The module contains the following ports:

Inputs: SSISelect: 2-bit signal that selects the clock source for each axis. '00' selects the MDT internal clock, and '01' selects the SSI clock. M_AX0_SSI_CLK: Input SSI clock signal for Axis0. M_AX1_SSI_CLK: Input SSI clock signal for Axis1. M_AX0_MDT_INT: Input MDT interrupt signal for Axis0. M_AX1_MDT_INT: Input MDT interrupt signal for Axis1.

Outputs: M_AX0_INT_CLK: Output internal clock signal for Axis0. M_AX1_INT_CLK: Output internal clock signal for Axis1.

The architecture MDTSSIRoute_arch describes the internal implementation of the MDTSSIRoute module. It includes two assignments that control the routing of clock signals based on the SSI select signal.

M_AX0_INT_CLK is assigned the value of M_AX0_MDT_INT when SSISelect(0) is '0', indicating that the MDT internal clock is selected for Axis0. Otherwise, it is assigned the value of M_AX0_SSI_CLK, indicating that the SSI clock is selected.

M_AX1_INT_CLK follows a similar assignment logic as M_AX0_INT_CLK, but for Axis1.

Overall, the MDTSSIRoute module provides the functionality to select the appropriate clock source for each axis based on the SSI select signal. It enables the routing of clock signals between the MDT internal clock and the SSI clock, allowing flexible control over the timing and synchronization of operations within the modular motion controller.

---

Entity Name: ControlIO

File: controlio.vhd

Description:

Control Board IO and LED Interface - The LED information is clocked out to 74HCT595 devices and is locked on the rising edge of the clock.

The module ControlIO is an entity in VHDL that describes the interface and behavior of a control module responsible for managing IO operations and LED status for two axes (Axis0 and Axis1). The module takes various input signals such as RESET, H1_CLKWR, SysClk, Enable, SynchedTick, intDATA, and several control signals for Axis0 and Axis1, and provides output signals for control and communication with other modules.

The primary purpose of the ControlIO module is to control the communication and configuration of LEDs and IO operations for Axis0 and Axis1. It includes a state machine that orchestrates the shifting of data in and out of shift registers, controls the timing and sequencing of operations, and manages the LED status and IO control signals.

The module contains the following ports:

Inputs:

RESET: Asynchronous reset signal.

H1_CLKWR: Clock signal for the module.

SysClk: System clock signal.

Enable: Enable signal for the module.

SynchedTick: Synchronous tick signal.

intDATA: Data input for IO operations.

Control signals for Axis0 and Axis1: These signals include signals for LED status read/write, IO read/write, and fault signals.

Outputs:

controlIoDataOut: Data output for control IO operations.

M_IO_OE: Output enable signal for the shift register controlling LEDs on axis modules.

M_IO_LOAD: Control signal to select input latch or shift register for IO operations.

M_IO_LATCH: Control signal to transfer data from shift register to outputs and latch inputs.

M_IO_CLK: Clock signal for input and output data through shift registers.

M_IO_DATAOut: Data output for IO operations.

M_ENABLE: Enable signals for Axis0 and Axis1.

QA0AxisFault: Output signals for Axis0 fault status.

QA1AxisFault: Output signals for Axis1 fault status.

The architecture ControlIO_arch describes the internal implementation of the ControlIO module. It includes several internal signals and components to manage the IO operations and LED status.

Key architecture components and processes:

ShiftOutRegister and ShiftInRegister are internal signals used to store the data being shifted in and out of the shift registers.

DataBufferOut and DataBufferIn are internal signals used to buffer the data coming in and out from the processor and the shift registers.

Count is a 4-bit signal used as a synchronous counter with count enable and asynchronous reset.

State is a 3-bit signal representing the current state of the state machine controlling the LED write sequence.

PowerUpLatch and StartStateMachine are signals used to control the state machine and manage the power-up sequence.

The StateMachine process controls the state transitions and operations of the LED write sequence based on the current state and input signals.

The M_LED_CLK process generates the output clock for the shift registers and increments the count for the state machine.

The module also includes various assignments and calculations for control signals, LED status, and fault signals based on the input and internal signals.

Overall, the ControlIO module provides the necessary functionality to manage the IO operations and LED status for Axis0 and Axis1. It uses a state machine to control the shifting of data in and out of shift registers, and it handles the timing and sequencing of operations to ensure proper communication and configuration of LEDs and IO signals.

---

| | |
|---|---|
| Entity Name: | MDTTopSimp |
| File: | MDTTopSimp.vhd |

Description:

MDT Interface - The MDT Interface provides a signal interface to PWM, Start/Stop, and SSI type transducers.

The MDTTopSimp module is a VHDL entity that represents the MDT interface in the RMC75E modular motion controller. This module provides a signal interface for PWM, Start/Stop, and SSI type transducers.

The module operates with three incoming clock signals: H1_CLK, H1_CLKWR, and H1_CLK90, all running at a rate of 60MHz. The H1_CLK signals are 90 degrees out of phase, allowing each clock to drive a counter when a start signal is received. At the end of the count cycle, the counters are added together.

The entity MDTTopSimp has the following ports:

Inputs: SysReset: Main system reset signal. H1_CLK: 60MHz clock for the MDT interface. H1_CLKWR: CPU clock for reads and writes. H1_CLK90: 60MHz clock with a 90-degree phase shift for MDT counters. SynchedTick60: Synchronized tick signal at 60MHz. intDATA: Input data signal (32 bits). PositionRead:

Signal indicating a position read operation. StatusRead: Signal indicating a status read operation. ParamWrite: Signal indicating a parameter write operation. M_RET_DATA: Input data signal for M_RET_DATA. SSISelect: Signal indicating the selection of SSI transducer.

Outputs: mdtSimpDataOut: Output data signal from the MDT (32 bits). M_INT_CLK: Output internal clock signal. SSI_DATA: Output SSI data signal.

The architecture MDTTopSimp_arch defines the internal implementation of the MDTTopSimp module. It includes various signals and processes to control the MDT interface.

The module includes a state machine (StateMachine) that controls the count sequence of the MDT counters. The state machine progresses through different states (s0, s1, s2, s3, s4, s5) based on different conditions and signals. The state machine handles the start and termination of the count cycle, detects return pulses, and sets various internal signals accordingly.

The module also includes counters (CountRA, Delay) for counting clock cycles, debounce flops for capturing rising and falling edges of the return pulse signals, and various control signals for enabling and disabling count cycles, delaying pulses, and setting data validity.

The position data is calculated and stored in the MDTPosition signal based on the count values and leading/trailing counts. The output data signal mdtSimpDataOut is set accordingly based on the input signals and the selection of SSI transducers.

Overall, the MDTTopSimp module provides the interface and control for the MDT in the RMC75E modular motion controller, allowing communication with PWM, Start/Stop, and SSI type transducers. It handles the counting sequence, detects pulses, and calculates position data based on the received signals.

---

Entity Name: Quad

File: quad.vhd

Description:

The Quad module

defines an entity with various input and output ports to handle multiple quadrature interfaces and axes.

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 the QuadXface component. 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.

---

Entity Name: QuadXface

File: QuadXface.vhd

Description:

The QuadXface module is a crucial component of the RMC75E modular motion controller. It is responsible for interfacing with quadrature encoder signals and performing various operations related to motion control.

The module features several input ports, including H1_CLKWR, SysClk, SynchedTick, intDATA, CountRead, LEDStatusRead, LEDStatusWrite, InputRead, HomeRead, Latch0Read, Latch1Read, Home, RegistrationX, RegistrationY, LineFault, A, B, and Index. These input ports facilitate the interaction between the module and external components, allowing for the transfer of control signals and data.

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 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.

The QuadXface module plays a vital role in the overall functionality of the RMC75E modular motion controller by providing essential features for quadrature encoding, counting, status management, edge detection, latch capture, and control configuration.

Entity Name: RAM128x16

File: RAM128x16.vhd

Description:

The RAM128x16 module is a component of the source code for the RMC75E modular motion controller. It serves as a 128x16-bit random access memory (RAM) module, capable of storing and retrieving data based on provided address and control signals.

Components:

Inputs:

- clk: The clock signal used for synchronous operations.
- we: The write enable signal that controls write operations.
- a: The address input signal used to specify the memory location to access.
- d: The data input signal containing the 16-bit data to be written into the specified memory location.

Outputs:

- The data output signal that provides the 16-bit data read from the specified memory location.

Architecture:

The architecture of the RAM128x16 module, named RAM128x16_arch, consists of the following components:

- ram_type: This is a defined type representing an array of 128 elements, where each element is a 16-bit std_logic_vector. It serves as the main storage for the RAM module.
- RAM: A signal of type ram_type, representing the actual memory storage. It is an array with 128 elements, each capable of storing a 16-bit value.
- read_a: A signal of type std_logic_vector(6 downto 0), used to hold the current address input for read operations.

The behavior of the RAM128x16 module is defined within a process block sensitive to the clk signal. The process handles both write and read operations based on the rising edge of the clock signal.

During a rising edge of the clock, the module checks if we (write enable) is asserted. If so, the module writes the 16-bit data d into the memory location specified by the address a.

The read_a signal is continuously updated with the current address a to ensure the correct data is read from the RAM during subsequent clock cycles.

Finally, the output signal o is assigned the value stored in the RAM at the memory location specified by read_a, providing the requested 16-bit data output.

The RAM128x16 module provides a reliable and efficient means of storing and retrieving data within the RMC75E modular motion controller, making it an integral component for various motion control operations.

---

Entity: rtdexpidled

File: rtdexpidled.vhd

Description:

The RtdExpIDLED module is a component of the source code for the RMC75E modular motion controller. It serves as a multiplexer and control logic for selecting and controlling the clock, latch, and load signals for an external LED module based on the state of the DiscoveryComplete signal.

Components:

Inputs:

- DiscoveryComplete: A control signal indicating the completion of the device discovery process.
- Exp_ID_CLK: The clock signal for the internal ID module.
- Exp_ID_LATCH: The latch signal for the internal ID module.
- Exp_ID_LOAD: The load signal for the internal ID module.
- ExpLEDOE: The output enable signal from the external LED module.
- ExpLEDLatch: The latch signal from the external LED module.
- ExpLEDClk: The clock signal from the external LED module.

Outputs:

- Exp_Mxd_ID_CLK: The selected clock signal for the ID module.
- Exp_Mxd_ID_LATCH: The selected latch signal for the ID module.
- Exp_Mxd_ID_LOAD: The selected load signal for the ID module.

Architecture:

The architecture of the RtdExpIDLED module, named RtdExpIDLED_arch, consists of a process block that controls the selection of clock, latch, and load signals based on the state of the DiscoveryComplete signal and the signals from the external LED module.

During the process, if the DiscoveryComplete signal is low (0), indicating that the device discovery process is not yet complete, the internal ID signals (Exp_ID_CLK, Exp_ID_LATCH, Exp_ID_LOAD) are selected and assigned to the corresponding output signals (Exp_Mxd_ID_CLK, Exp_Mxd_ID_LATCH, Exp_Mxd_ID_LOAD).

However, if the DiscoveryComplete signal is high (1), indicating that the device discovery process is complete, the signals from the external LED module (ExpLEDClk, ExpLEDLatch, ExpLEDOE) are selected and assigned to the corresponding output signals (Exp_Mxd_ID_CLK, Exp_Mxd_ID_LATCH, Exp_Mxd_ID_LOAD).

The RtdExpIDLED module provides the necessary logic to control the selection of clock, latch, and load signals based on the state of the DiscoveryComplete signal, allowing seamless integration of the external LED module into the RMC75E modular motion controller.

---

Module Name: Serial2Parallel

File: serial2parallel.vhd

Description:

The Serial2Parallel module is a component of the source code for the RMC75E modular motion controller. It serves as a serial-to-parallel converter and data distributor for various input signals. The module takes serial input data and converts it into parallel format for multiple output channels based on the control signals.

Components:

Inputs:

- SysClk: The system clock signal.
- SynchedTick: The synchronized tick signal.
- CtrlAxisData: Two-bit control axis data input.
- ExpA_DATA: Eight-bit data input for expansion module A.
- Serial2ParallelEN: The enable signal for the Serial2Parallel module.
- Serial2ParallelCLR: The clear signal for resetting the internal data registers.
- S2P_Addr: Four-bit address input for selecting the output channel.

Outputs:

- S2P_Data: Sixteen-bit parallel data output.
-

Architecture:

The architecture of the Serial2Parallel module, named Serial2Parallel_arch, consists of internal signals and a process block for data conversion and distribution.

Internal Signals:

- S2P_M0Ch0_Data: Sixteen-bit data for Module 0, Channel 0.
- S2P_M0Ch1_Data: Sixteen-bit data for Module 0, Channel 1.
- S2P_M1Ch0_Data: Sixteen-bit data for Module 1, Channel 0.

- S2P_M1Ch1_Data: Sixteen-bit data for Module 1, Channel 1.
- S2P_M2Ch0_Data: Sixteen-bit data for Module 2, Channel 0.
- S2P_M2Ch1_Data: Sixteen-bit data for Module 2, Channel 1.
- S2P_M3Ch0_Data: Sixteen-bit data for Module 3, Channel 0.
- S2P_M3Ch1_Data: Sixteen-bit data for Module 3, Channel 1.
- S2P_CtrlAxis0_Data: Sixteen-bit data for Control Axis 0.
- S2P_CtrlAxis1_Data: Sixteen-bit data for Control Axis 1.

The Serial2Parallel module includes a process block that performs the following operations based on the rising edge of the SysClk signal:

- When SynchedTick or Serial2ParallelCLR is high, indicating a reset condition, all internal data registers are cleared to 0.
- When Serial2ParallelEN is high, the module enables the data conversion and distribution process. The input data is shifted into the respective internal data registers for each output channel, expanding the serial input data into 16-bit parallel words.
- The S2P_Data output is determined based on the S2P_Addr input. The selected output channel's data is assigned to the S2P_Data output.

The Serial2Parallel module provides the functionality of converting serial input data into parallel format and distributing it to the appropriate output channels based on control signals. This module facilitates efficient data handling and distribution within the RMC75E modular motion controller.