*Project: RMC75E FPGA TEST BENCH*

*Module: serial\_mem.vhd*

*Author: Satchel Hamilton*

*Company: Delta Motion*

*Date: 7/13/2023*

*Last updated: July 13, 2023*

Contents

[High Level 1](#_Toc140670735)

[Low level: 3](#_Toc140670736)

[Simulation: 4](#_Toc140670737)

# High Level

The `SerialMemoryInterface` module is designed as a serial memory interface for the RMC75E modular motion controller built by Delta Motion. The module provides communication with an external serial EEPROM, specifically the AT24C01A device.

The **SerialMemoryInterface** component converts data from parallel to serial format before sending it to the EEPROM (Electrically Erasable Programmable Read-Only Memory). This conversion is necessary because many EEPROM devices communicate using a serial protocol, which means they expect data to be sent one bit at a time.

Here's how the data conversion process works:

1. **Data Parallelization**: The external system provides data in parallel format through the **intDATA** signal, which is a 32-bit wide vector. This data needs to be written to the EEPROM.
2. **Serialization**: The **SerialMemoryInterface** converts this parallel data into a serial stream. It does so by sequentially sending each bit of the data to the EEPROM on the **intSerialMemoryDataOut** signal. The **intSerialMemoryClock** signal controls the timing of the data transmission.
3. **Start and Stop Bits**: Before sending the actual data, the interface generates start and stop bits. A start bit (logic '0') indicates the beginning of the data transmission, while a stop bit (logic '1') marks the end of the data.
4. **Shift Register**: The component uses a shift register (**SerialDataOutput**) to hold the data temporarily while serializing it. The data is shifted out bit by bit during the communication process.
5. **ACK Handling**: After sending the data, the component waits for an acknowledgment signal (**ACK**) from the EEPROM. This ACK signal confirms that the EEPROM has received the data correctly.

For a write operation, the state machine follows a specific sequence (Start, Device Address, Memory Address, Data) to write the data into the EEPROM. The ACK handling is essential to ensure successful data transmission.

The component also supports reading data from the EEPROM using similar serialization and parallelization processes. However, in the read operation, the ACK handling is not required as data is being read from the EEPROM rather than written to it.

Overall, the **SerialMemoryInterface** is responsible for managing the communication protocol and data format conversion between the external parallel data bus and the EEPROM's serial interface.

- Inputs:

- `SysReset`: System Reset or PLL not locked signal.

- `H1\_CLK`: 60MHz clock signal.

- `SysClk`: 30MHz system clock signal.

- `SlowEnable`: Signal to enable slow operations.

- `intDATA`: Input data to be written to the serial memory.

- `SerialMemXfaceWrite`: Signal to initiate a write operation.

- `SerialMemoryDataIn`: Input data from the serial memory.

- Outputs:

- `serialMemDataOut`: Output data from the serial memory.

- `SerialMemDataOut`: Output data from the serial memory (connected to the `serialMemDataOut` internally).

- `SerialMemoryDataControl`: Control signal for the serial memory data.

- `SerialMemoryClk`: Clock signal for the serial memory.

- `Exp0SerialSelect`, `Exp1SerialSelect`, `Exp2SerialSelect`, `Exp3SerialSelect`: Selection signals for expansion slots.

- `EEPROMAccessFlag`: Flag indicating access to the EEPROM.

- `M\_SPROM\_CLK`: Clock signal for the serial EEPROM module.

- `M\_SPROM\_DATA`: Bidirectional data signal for the serial EEPROM module.

The module operates based on a state machine that controls serial communication with the AT24C01A EEPROM. The state machine progresses through different states to perform operations such as writing to and reading from the EEPROM.

The main states of the state machine are as follows:

1. IdleState: The initial state where the module waits for a write or read operation to be triggered.

2. SignalStartState: Sends a start condition to initiate the communication.

3. DeviceAddrState: Loads the device address into the shift register for transmission.

4. CheckDeviceAddrACKState: Checks the acknowledgment (ACK) received after sending the device address.

5. MemAddrState: Loads the memory address into the shift register for transmission.

6. CheckMemoryAddrACKState: Checks the ACK received after sending the memory address.

7. OperationTypeState: Determines the operation type (write or read) based on the WriteFlag signal.

8. WriteState: Transmits the data to be written to the EEPROM.

9. WriteAckState: Checks the ACK received after writing data.

10. ReadState: Reads data from the EEPROM by shifting it out bit by bit.

11. ReadNoACKState: Sends a NACK (no acknowledgment) after reading data.

12. StopState: Sends a stop condition to end the communication.

13. ClearState: Clears internal flags and prepares for the next operation.

The module generates the necessary control signals and manages the data transfer between the motion controller and the serial memory. It also handles error conditions and keeps track of operation faults.

# Low level:

The module has the following input ports:

- `SysReset`: System reset signal or PLL not locked indication.

- `H1\_CLK`: 60MHz clock signal.

- `SysClk`: 30MHz system clock signal.

- `SlowEnable`: Signal to enable slow operation mode.

- `intDATA`: 32-bit input data from the motion controller.

- `SerialMemXfaceWrite`: Signal indicating a write operation to the serial memory device.

- `SerialMemoryDataIn`: Input data from the serial memory device.

- `M\_SPROM\_DATA`: Bidirectional data signal for communication with the Serial EEPROM.

And the following output ports:

- `serialMemDataOut`: 32-bit data output for the motion controller.

- `SerialMemoryDataOut`: Output data signal to the serial memory device.

- `SerialMemoryDataControl`: Output control signal for the serial memory device.

- `SerialMemoryClk`: Output clock signal for the serial memory device.

- `Exp0SerialSelect`, `Exp1SerialSelect`, `Exp2SerialSelect`, `Exp3SerialSelect`: Output select signals for expansion modules.

- `EEPROMAccessFlag`: Output signal indicating access to the Serial EEPROM.

- `M\_SPROM\_CLK`: Output clock signal for the Serial EEPROM.

- `M\_SPROM\_DATA`: Bidirectional data signal for communication with the Serial EEPROM.

At the lowest and most granular level, the **SerialMemoryInterface** module operates using a clocked state machine to control the data transfer between the processor (CPU) and the Serial EEPROM device. Let's delve into the granular details of how the module works:

1. Clock Generation:
   * The module generates two clock signals: **SerialMemoryClk** and **intSerialMemoryClock**.
   * The **SerialMemoryClk** is used for clocking data in and out of the Serial EEPROM device.
   * The **intSerialMemoryClock** is an internally generated clock used for timing within the module.
   * The **SerialMemoryClockEnable** signal controls the generation of the **intSerialMemoryClock** and ensures the clock period does not exceed 100 kHz.
2. State Machine:
   * The module employs a state machine (**StateMachine**) to manage the communication process with the Serial EEPROM device.
   * The state machine sequentially moves through different states to carry out read and write operations.
3. Data Transfer:
   * The **SerialDataOutput** and **SerialDataInput** signals are used to buffer data being transmitted to and received from the Serial EEPROM device.
   * The **SerialDataOutput** signal contains the data to be sent to the EEPROM during write operations, while **SerialDataInput** holds the data received during read operations.
4. Communication Protocol:
   * The module follows a specific communication protocol to interact with the Serial EEPROM device.
   * Communication begins with a "Start" condition, followed by sending the device address, memory address, and data during write operations.
   * For read operations, the module sends the device address and memory address, followed by a repeated "Start" condition to initiate read data retrieval.
5. Data Shift Register:
   * The module utilizes a shift register to serialize and deserialize data for communication with the Serial EEPROM device.
   * Various control signals (**ShiftEnable**, **LoadDeviceAddr**, **LoadMemAddr**, and **LoadWriteData**) control the loading and shifting of data in and out of the shift register.
6. Error Handling:
   * The module includes logic to handle cases where the Serial EEPROM device does not acknowledge communication attempts (NO ACK response).
   * The **OperationFaultFlag** and **OperationFaultCount** signals are used to detect and track communication failures.
7. Control Signals:
   * The module uses several control signals (**WriteFlag**, **ReadFlag**, **ACK**, **StartStopBit**, etc.) to coordinate the various stages of data transfer and communication with the Serial EEPROM device.
8. Tri-state Buffering:
   * To enable bidirectional communication with the Serial EEPROM device, the module uses tri-state buffers for **M\_SPROM\_DATA**, allowing it to drive the data line when needed and relinquish control when not required.
9. Interface Selection:
   * The module uses various control signals (**ControlSerialSelect**, **Exp0SerialSelect**, etc.) to determine which external module interface to interact with during EEPROM access.

Overall, the **SerialMemoryInterface** module orchestrates the precise timing and control signals required for reliable data transfer with a Serial EEPROM device. It uses the state machine to guide the communication protocol and ensures the correct sequence of operations for both read and write functionalities.

The module is implemented using an architecture called `SerialMemoryInterface\_arch`. It contains several constants, signals, and processes that control the state machine and data flow between the motion controller and the serial memory device.

The architecture defines various constant values used for device addresses, clock terminal counts, and selection addresses for expansion modules. These constants are converted to the appropriate vector types using the `To\_StdLogicVector` function.

The module uses a state machine (`StateMachine`) to control serial communication with the Serial EEPROM. The state machine has several states, including IdleState, SignalStartState, DeviceAddrState, CheckDeviceAddrACKState, MemAddrState, CheckMemoryAddrACKState, OperationTypeState, WriteState, WriteAckState, ReadState, ReadNoACKState, StopState, and ClearState.

The architecture includes processes that handle clock generation, write and read flags, module and memory addresses, data buffering, ACK (acknowledgment) signal handling, and state transitions.

The module also generates control signals (`intSerialMemoryDataControl`, `intSerialMemoryClock`, `intSerialMemoryDataOut`) and manages the selection signals for the expansion modules (`Exp0SerialSelect`, `Exp1SerialSelect`, `Exp2SerialSelect`, `Exp3SerialSelect`).

Additionally, there are processes that handle the shifting of data bits, capturing input data, counting shifted bits, handling operation faults, and generating the final output signals (`serialMemDataOut`, `intEEPROMAccessFlag`, `intOperationFaultFlagInput`, `OperationFaultFlag`).

Overall, this module provides an interface for the RMC75E motion controller to communicate with a Serial EEPROM, enabling read and write operations and managing various control signals and data transfers.

## Simulation:

To ensure that SerialMemoryDataOut shows relevant data and is not undefined at the end of the 100 us simulation, the following conditions need to be met:

1. The system reset signal (SysReset) should be deasserted ('0') after the appropriate reset period to initialize the system.
2. The serial memory interface write signal (SerialMemXfaceWrite) should be asserted ('1') during the desired write operation.
3. The module address (intModuleAddress) should be set to the appropriate value for the desired memory module.
4. The memory address (MemoryAddress) should be set to the desired address where data needs to be written or read from.
5. The data to be written (DataBuffer) should be loaded into the shift register when performing a write operation.
6. The shift enable signal (ShiftEnable) should be activated to allow data shifting during the write or read operation.
7. The appropriate clock signals (SysClk and H1\_CLK) should be provided with the correct timing and frequency to synchronize the operations.
8. The slow enable signal (SlowEnable) should be asserted ('1') to enable the clock division and control the speed of the serial interface.
9. The simulation should run for the desired duration of 100 us to ensure sufficient time for the operations to complete.
10. By ensuring these conditions are met, the SerialMemoryDataOut signal will show relevant data and not be undefined at the end of the 100 us simulation.