Project Status Report – RMC75E Test Bench

Date: August 4th, 2023

Subject: Progress Update

Current Phase: Refining & Expanding existing test units and documentation.

Challenges: One major challenge I was dealing with this week was getting the simulation for the serial memory interface module to behave correctly. Initially the issue was that the state machine was operating in such a way that it seemed not to allow the full data transmission on M_SPROM_DATA bidirectional data line. It would basically force the line to go to high impedance when there was still data to transmit.

- On that note, the state machine logic in the serial memory interface looks like it could benefit from some consolidation.
- There seems to be a lack of logical separation / organization that makes it very hard to follow, and (as far as I can tell with my limited experience) its logic could definitely be optimized.
- Many logical statements are repeated.
- To wit, an overabundance of states (WriteACKstate, CheckMemoryAddrACKState, CheckDeviceAddrACKState) seemed to be setting the intSerialMemoryDataControl signal low, which was causing the M_SPROM_DATA output signal to be truncated.
- A lot of the logic in the state machine feels like it suffers from a case of “too many cooks in the kitchen”, that is, many different process blocks are trying to control the same signals.

Removing that logic on lines 321, 351, 387 has allowed the signal to output in full.

The **M_SPROM_DATA** signal is part of the **SerialMemoryInterface** entity, which is the interface to communicate with a serial EEPROM (Electrically Erasable Programmable Read-Only Memory) device via the I2C protocol, specifically the AT24C01A.

The purpose of the **M_SPROM_DATA** signal is to provide bidirectional data communication with the serial EEPROM module when the interface is selected for communication. This signal is declared as **inout**, which means it can be used as both an input and an output. It allows data to be sent to the serial EEPROM for write operations and also receives data from the serial EEPROM during read operations.

The specific data transmitted on this line depends on the operation being performed (read or write) and the address and data being sent. The data format and protocol are controlled by the state machine (**StateMachine**) in the architecture. The state machine manages the entire communication sequence, including address transmission, read/write operations, and handling acknowledgment signals.

Here's a summary of the main states and operations of the state machine in the architecture:

1. SignalStartState: This state is responsible for initiating communication with the serial EEPROM by sending a Start condition on the bus.

2. DeviceAddrState: In this state, the device address of the serial EEPROM is loaded into the shift register to prepare for transmission.
3. CheckDeviceAddrACKState: After sending the device address, the state machine checks for acknowledgment (ACK) from the serial EEPROM. If ACK is received, it proceeds to the next state; otherwise, it may retry or signal an operation fault.
4. MemAddrState: In this state, the memory address for read/write operations is loaded into the shift register to prepare for transmission.
5. CheckMemoryAddrACKState: Similar to the device address check, this state checks for acknowledgment (ACK) for the memory address sent.
6. OperationTypeState: This state determines the type of operation to be performed (read or write).
7. WriteState: In write operations, this state is responsible for transmitting the data to be written to the serial EEPROM.
8. WriteAckState: After sending the data, this state checks for acknowledgment (ACK) from the serial EEPROM. If ACK is received, it proceeds to the StopState; otherwise, it may retry or signal an operation fault.
9. ReadState: In read operations, this state enables the counter to clock, indicating the number of bits to be read.
10. ReadNoACKState: In read operations, this state outputs a '1' during the ACK phase.
11. StopState: The state machine signals a Stop condition to end the communication sequence.
12. ClearState: This state clears the FLAG_CLR signal, indicating the end of the operation.

The **M_SPROM_DATA** signal will carry different data during each of these states to perform the required read or write operations with the serial EEPROM.

Here we can see a bit-by-bit breakdown of the data contained in the M_SPROM_DATA signal:

Bit 31: Operation Fault Flag - Indicates if there was an operation fault during EEPROM communication.

Bit 30: Serial Data Input Bit 7 - Represents the most significant bit of the data input during read operations.

Bit 29: Serial Data Input Bit 6 - Represents the 6th bit of the data input during read operations.

...

Bit 23: Serial Data Input Bit 0 - Represents the least significant bit of the data input during read operations.

Bit 22: Operation Fault Flag Input - Input to check if there was an operation fault during EEPROM communication.

Bit 21: Inc Operation Fault Count - Signal to increment the operation fault counter during communication.

Bit 20: Second Pass Read - Used by the Read operation to loop through the state machine multiple times.

Bit 19: Start/Stop Bit - Output to signal a Start or Stop condition during communication.

Bit 18: Shift Enable - Signal to enable the shifting of data during communication.

Bit 17: Serial Memory Data Control - Signal to control the direction of data transmission (input/output).

Bit 16: EEPROM Access Flag - Output to indicate control over the data line to the EEPROM.

Bit 15: Expansion Module Select Address Bit 2 - Used to select the appropriate expansion module.

Bit 14: Expansion Module Select Address Bit 1 - Used to select the appropriate expansion module.

Bit 13: Expansion Module Select Address Bit 0 - Used to select the appropriate expansion module.

Bit 12: Control Module Select - Signal to select the control module.

Bit 11: Memory Address Bit 5 - Represents the 6th bit of the memory address for read/write operations.

...

Bit 6: Memory Address Bit 0 - Represents the least significant bit of the memory address for read/write operations.

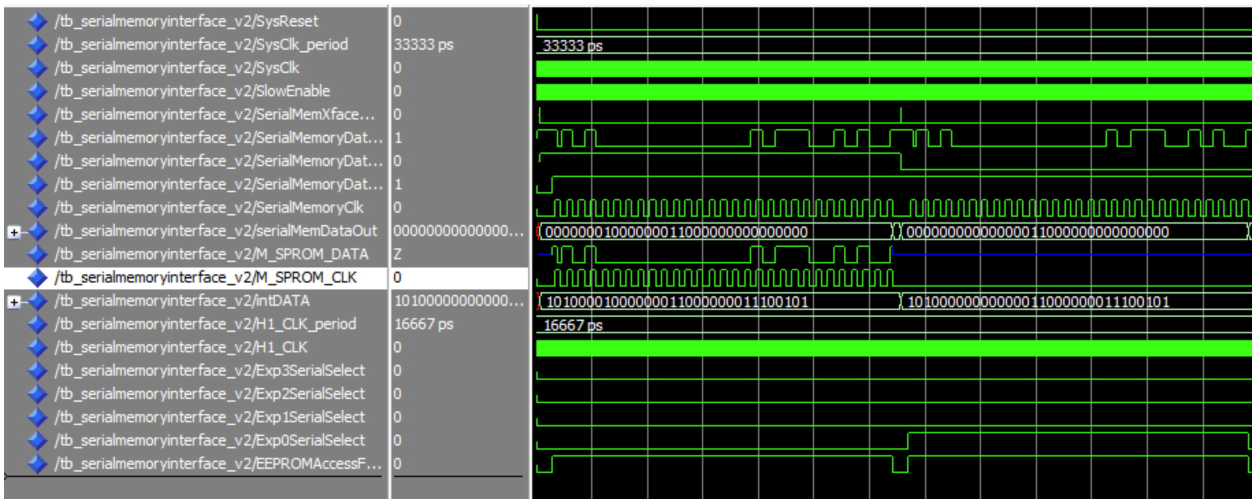
Bit 5: Data Buffer Bit 7 - Represents the most significant bit of the data buffer for write operations.

...

Bit 0: Data Buffer Bit 0 - Represents the least significant bit of the data buffer for write operations.

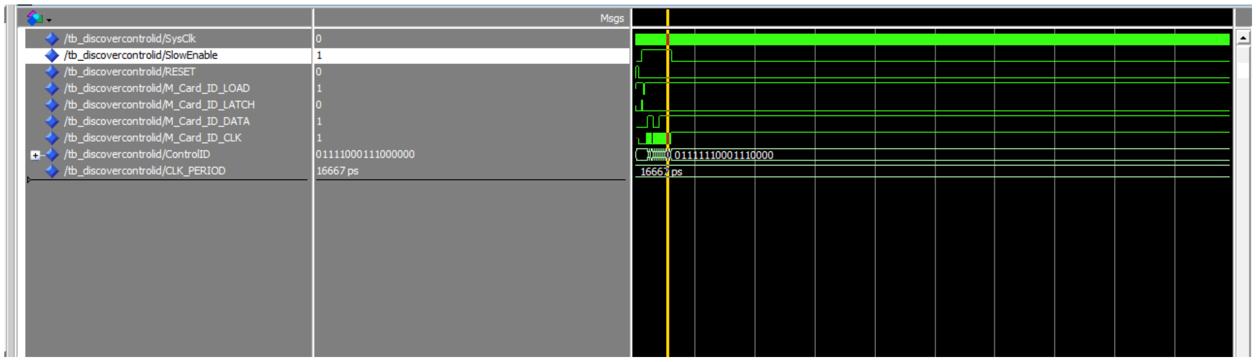
Work to date: This week I was able to get the serial memory module simulating correctly as far as I can tell. I will need to go over it with David to ensure that it is behaving the way it is expected to. I also refined the expansion signal routing module and the modules that handle control ID and expansion module discovery. I think overall the test bench is looking good. All test units are simulating well and displaying expected behavior.

Fig. 1 – Serial Memory Interface



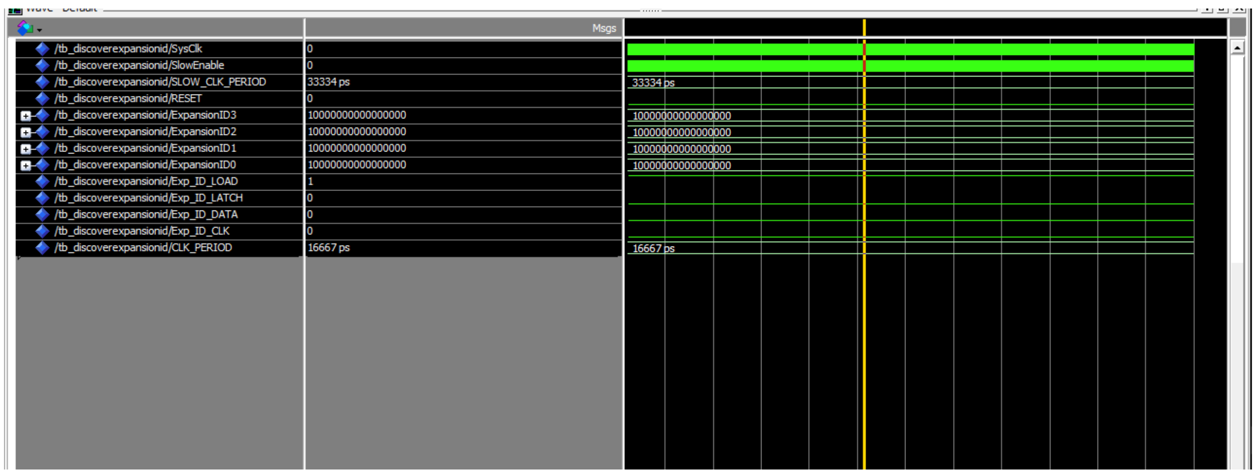
```
/tb_serialmemoryinterface_v2/SysReset                0
/tb_serialmemoryinterface_v2/SysClk_period           33333 ps
/tb_serialmemoryinterface_v2/SysClk                  0
/tb_serialmemoryinterface_v2/SlowEnable              0
/tb_serialmemoryinterface_v2/SerialMemXface...       0
/tb_serialmemoryinterface_v2/SerialMemoryDat...      1
/tb_serialmemoryinterface_v2/SerialMemoryDat...      0
/tb_serialmemoryinterface_v2/SerialMemoryDat...      1
/tb_serialmemoryinterface_v2/SerialMemoryClk         0
/tb_serialmemoryinterface_v2/serialMemDataOut        00000000000000...
/tb_serialmemoryinterface_v2/M_SPROM_DATA            Z
/tb_serialmemoryinterface_v2/M_SPROM_CLK             0
/tb_serialmemoryinterface_v2/intDATA                 10100000000000...
/tb_serialmemoryinterface_v2/H1_CLK_period           16667 ps
/tb_serialmemoryinterface_v2/H1_CLK                  0
/tb_serialmemoryinterface_v2/Exp3SerialSelect        0
/tb_serialmemoryinterface_v2/Exp2SerialSelect        0
/tb_serialmemoryinterface_v2/Exp1SerialSelect        0
/tb_serialmemoryinterface_v2/Exp0SerialSelect        0
/tb_serialmemoryinterface_v2/EEPROMAccessF...        0
```

Fig 2 - ControlID Discovery



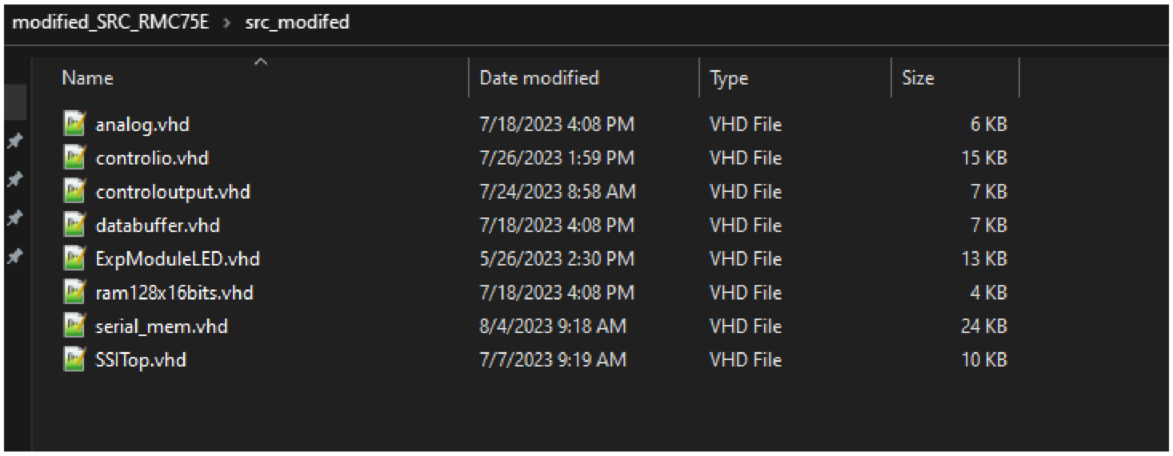
```
Msgs
/tb_discovercontrolid/SysClk           0
/tb_discovercontrolid/SlowEnable       1
/tb_discovercontrolid/RESET            0
/tb_discovercontrolid/M_Card_ID_LOAD   1
/tb_discovercontrolid/M_Card_ID_LATCH  0
/tb_discovercontrolid/M_Card_ID_DATA   1
/tb_discovercontrolid/M_Card_ID_CLK    1
/tb_discovercontrolid/ControlID        01111000111000000
/tb_discovercontrolid/CLK_PERIOD       16667 ps
```

Fig 3 - ExpansionID Discovery



```
Msgs
/tb_discoverexpansionid/SysClk            0
/tb_discoverexpansionid/SlowEnable        0
/tb_discoverexpansionid/SLOW_CLK_PERIOD   33334 ps
/tb_discoverexpansionid/RESET             0
/tb_discoverexpansionid/ExpansionID3      10000000000000000
/tb_discoverexpansionid/ExpansionID2      10000000000000000
/tb_discoverexpansionid/ExpansionID1      10000000000000000
/tb_discoverexpansionid/ExpansionID0      10000000000000000
/tb_discoverexpansionid/Exp_ID_LOAD       1
/tb_discoverexpansionid/Exp_ID_LATCH      0
/tb_discoverexpansionid/Exp_ID_DATA       0
/tb_discoverexpansionid/Exp_ID_CLK        0
/tb_discoverexpansionid/CLK_PERIOD        16667 ps
```

Fig 4 – Source Modifications


modified_SRC_RMC75E › src_modifed

Name	Date modified	Type	Size
analog.vhd	7/18/2023 4:08 PM	VHD File	6 KB
controlio.vhd	7/26/2023 1:59 PM	VHD File	15 KB
controloutput.vhd	7/24/2023 8:58 AM	VHD File	7 KB
databuffer.vhd	7/18/2023 4:08 PM	VHD File	7 KB
ExpModuleLED.vhd	5/26/2023 2:30 PM	VHD File	13 KB
ram128x16bits.vhd	7/18/2023 4:08 PM	VHD File	4 KB
serial_mem.vhd	8/4/2023 9:18 AM	VHD File	24 KB
SSITop.vhd	7/7/2023 9:19 AM	VHD File	10 KB

Next Steps: Continue refining the test units according to David's instructions. Continue refining and adding to the documentation of the source code. Make notes on recommendations for improving the source code. Add line comments to the source code.