I2C Multi-Bus Controller

Test Plan
Issue 4

Group 6

Blagojevic (11904645)
Kenbeek (12229949)
Scharwitzl (11909460)

Contents

1 Document History 2
2 Reference Documents 2
3 Overview 3
4 Test Plan 4
4.1 Clock and Reset 4
4.2 Interface Requirements 5
4.2.1 Control Interface 5
4.2.2 I2C Interfaces 7
4.3 I2C Communication Protocol (Read/Write) 10
4.3.1 Communication Header 10
4.3.2 Write 11
4.3.3 Read 12
4.3.4 End of Transfer 13
4.4 Interrupt 14

1 Document History

Issue Number	Issue Date	Changes
1	29. April 2024	Initial issue
2	05. June 2024	- Clarified how to test system clock in T-001 - Clarified pre-test conditions for T-002 and T-003 - Clarified the clock range to test for T-024 to T-027 - Clarified the values which should be tested for T-029, T-039, T-032, T-036, T-045
3	10. June 2024	- Clarified data values for T-015 and T-016 and add T050, T-051, T052 and T-053 - Remove T-004, T-005, T-006 and T-007 - Add T-055, T-056, T-057 and T-058 - Specify the data words in T-011 and T-013 - Specify which bits are from interest in T-012, T-014, T-009 and T-010
4	16. June 2024	Added Coverage Links

2 Reference Documents

[SPEC]	-	I2C Multi-Bus Controller, Specification Document

3 Overview

The purpose of this test plan is to ensure the proper functioning and reliability of the I2C-Controller (I2CC). This test plan aims to validate the functionality of the I2C controller across various scenarios and conditions to guarantee its performance under normal operation as well as under stress conditions. This test plan divides the test cases into four categories.

In Clock and Reset we define test cases to verify the reset behaviour of the I2CC. The basic AVMM and I2C interface specifications are verified in the Interface Requirements section. The correct I2C communication protocol for a read and write transfer is verified in the section Communication Protocol. Lastly, the test cases of the section Interrupt check if interrupts are issued at the correct time.

When possible we aim to implement a covergroup test strategy.

As described in the specification and after further discussion a concurrent read and write access is excluded.

When describing a bit vector an “*” generally means that we do not care about the value at this specific bit.

4 Test Plan

4.1 Clock and Reset

ID	Title	Description	Design Requirements	Coverage Link
T-002	Reset	First change the default values of all registers, then if reset is set to 1 for at least one system clock cycle, all registers are reset to their specified reset values.	*I2C-003, I2C-004*	*tb_reset.stimuli_p(30-35)*
T-003	Soft reset	First change the default values of all registers, then if ControlReg.RST is set to 1, then every register should be reset to their specified reset values.	I2C-053, I2C-054	*tb_reset.stimuli_p(45-50)*

4.2 Interface Requirements

4.2.1 Control Interface

ID	Title	Description	Design Requirements	Coverage Link
T-008	AVMM sync	Verify that every change of the readdata signal must be synchronous to the AVMM clock.	*I2C-013*	*avmm_vu.check_sync(74)*
T-009	AVMM ControlReg Wr	If address is set to 0x0, a 32-bit data word d is applied to writedata, byteenable is set to 0xF and write is set to 1, then d should be in the ControlReg, where we do not care about the values of the bits 2, 3, 10-15 and 30.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-011, I2C-012*	*tb_avmm.stimuli_p(119)*
T-010	AVMM ControlReg Rd	If address is set to 0x0, byteenable is set to 0xF, read is set to 1 and d is the content of ControlReg, then after one AVMM-clock-cycle readdata should be equal to d, where we do not care about the values of the bits 2, 3, 10-15 and 30.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(119)*
T-011	AVMM StatusReg Wr	If address is set to 0x1, the 32-bit word 0x55555555 is applied to writedata, byteenable is set to 0xF and write is set to 1, then “*****************************101” should be in the StatusReg.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-011, I2C-012*	*tb_avmm.stimuli_p(99)*
T-012	AVMM StatusReg Rd	If address is set to 0x1, byteenable is set to 0xF, read is set to 1 and d is the content of StatusReg, then after one AVMM-clock-cycle the lower 3 bit of readdata should be equal to the lower 3 bit of d.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(99)*

T-013	AVMM BusEnReg Wr	If address is set to 0x2, the 32-bit data word 0x55555555 is applied to writedata, byteenable is set to 0xF and write is set to 1, then 0x*******5 should be in the BusEnReg.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-011, I2C-012*	*tb_avmm.stimuli_p(109)*
T-014	AVMM BusEnReg Rd	If address is set to 0x2, byteenable is set to 0xF, read is set to 1 and d is the content of BusEnReg, then after one AVMM-clock-cycle the lower 4 bit of readdata should be equal to the lower 4 bit of d.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(109)*
T-015	AVMM DataReg0 Wr 0xA	If address is set to 0x10, 0xAAAAAAAA applied to writedata, byteenable is set to 0xF and write is set to 1, then 0xAAAAAAAA should be in the DataReg0.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-011, I2C-012*	*tb_avmm.stimuli_p(26)*
T-051	AVMM DataReg0 Wr 0x5	If address is set to 0x10, 0x555555555 applied to writedata, byteenable is set to 0xF and write is set to 1, then 0x555555555 should be in the DataReg0.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-011, I2C-012*	*tb_avmm.stimuli_p(34)*
T-016	AVMM DataReg0 Rd 0xA	If address is set to 0x10, byteenable is set to 0xF, read is set to 1 and 0xAAAAAAAA is the content of DataReg0, then after one AVMM-clock-cycle readdata should be equal to 0xAAAAAAAA.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(26)*
T-052	AVMM DataReg0 Rd 0x5	If address is set to 0x10, byteenable is set to 0xF, read is set to 1 and 0x555555555 is the content of DataReg0, then after one AVMM-clock-cycle readdata should be equal to 0x555555555.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(34)*
T-017	AVMM DataReg15 Wr 0xA	If address is set to 0x1F, 0xAAAAAAAA is applied to writedata, byteenable is set to 0xF and write is set to 1, then 0xAAAAAAAA should be in the DataReg15.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-011, I2C-012*	*tb_avmm.stimuli_p(42)*

T-053	AVMM DataReg15 Wr 0x5	If address is set to 0x1F, 0x555555555 is applied to writedata, byteenable is set to 0xF and write is set to 1, then 0x555555555 should be in the DataReg15.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-011, I2C-012*	*tb_avmm.stimuli_p(50)*
T-018	AVMM DataReg15 Rd 0xA	If address is set to 0x1F, byteenable is set to 0xF, read is set to 1 and 0xAAAAAAAA is the content of DataReg15, then after one AVMM-clock-cycle readdata should be equal to 0xAAAAAAAA.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(42)*
T-054	AVMM DataReg15 Rd 0x5	If address is set to 0x1F, byteenable is set to 0xF, read is set to 1 and 0x555555555 is the content of DataReg15, then after one AVMM-clock-cycle readdata should be equal to 0x555555555.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(50)*
T-055	AVMM Byte_En 1	If a write was performed on the address 0x10, with the byteenable 0x1 and the writedata was 32-bit data word d, then when reading from address 0x10 only the lowest byte should have changed.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(61)*
T-056	AVMM Byte_En 2	If a write was performed on the address 0x10, with the byteenable 0x2 and the writedata was 32-bit data word d, then when reading from address 0x10 only the lowest byte should have changed.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(70)*
T-057	AVMM Byte_En 3	If a write was performed on the address 0x10, with the byteenable 0x3 and the writedata was 32-bit data word d, then when reading from address 0x10 only the lowest byte should have changed.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(79)*

T-058	AVMM Byte_En 4	If a write was performed on the address 0x10, with the byteenable 0x4 and the writedata was 32-bit data word d, then when reading from address 0x10 only the lowest byte should have changed.	*I2C-006, I2C-007, I2C-008, I2C-009, I2C-010, I2C-012*	*tb_avmm.stimuli_p(88)*

4.2.2 I2C Interfaces

ID	Title	Description	Design Requirements	Coverage Link
T-019	Multiple I2C interfaces	Verify there are 4 I2C bus interfaces available	*I2C-014*	*tb_i2c_interfaces.stimuli_p(24)*
T-020	Bus signals presence	For each of the I2C bus interfaces, verify the existence of a separate SCL and SDA signal	*I2C-014, I2C-015*	*tb_i2c_interfaces_read, tb_i2c_interfaces_write*
T-021	Idle bus	For each of the I2C bus interfaces, the SCL/SDA signals shall be high-Z when there is no ongoing bus transfer	*I2C-014, I2C-017*	*i2c_vu.WaitForStart*
T-022	START conditions	For each of the I2C bus interfaces, verify that the controller can send a START condition and repeated START conditions with the correct timing requirements for t_{LOW}, t_{HIGH}, $t_{HD;STA}$, and $t_{SU;STA}$	*I2C-014, I2C-018, I2C-021, I2C-022, I2C-023, I2C-024, I2C-025*	*Timing requirements in i2c_vu.timing_p*
T-023	STOP condition	For each of the I2C bus interfaces, verify that the controller can send a STOP condition with the correct timing requirements for t_{HIGH}, and $t_{SU;STO}$	*I2C-014, I2C-018, I2C-021, I2C-023, I2C-026*	*Timing requirements in i2c_vu.timing_p*

T-024	Sending slow data	For each of the I2C bus interfaces, verify that the controller can send a START condition, followed by bit data and at least one additional START conditions, and finally a STOP condition to a target with timing requirements t_{LOW}, t_{HIGH}, $t_{HD;STA}$, $t_{SU;STA}$, $t_{SU;STO}$, and $t_{SU;DAT}$ while maintaining a bus speed of 10 kbit/s. Repeat this at least once while meeting timing requirement t_{BUF}. This test should be performed at a 25 MHz clock frequency and 125 MHz.	*I2C-014, I2C-016, I2C-018, I2C-019, I2C-020, I2C-021, I2C-022, I2C-023, I2C-024, I2C-025, I2C-026, I2C-027, I2C-028*	*Timing requirements in i2c_vu.timing_p*
T-025	Sending fast data	For each of the I2C bus interfaces, verify that the controller can send a START condition, followed by bit data and at least one additional START conditions, and finally a STOP condition to a target with timing requirements t_{LOW}, t_{HIGH}, $t_{HD;STA}$, $t_{SU;STA}$, $t_{SU;STO}$, and $t_{SU;DAT}$ while maintaining a bus speed of 400 kbit/s. Repeat this at least once while meeting timing requirement t_{BUF}. This test should be performed at a 25 MHz clock frequency and 125 MHz.	*I2C-014, I2C-016, I2C-018, I2C-019, I2C-020, I2C-021, I2C-022, I2C-023, I2C-024, I2C-025, I2C-026, I2C-027, I2C-028*	*Timing requirements in i2c_vu.timing_p*
T-026	Receiving slow data	For each of the I2C bus interfaces, verify that the controller can receive a START condition, followed by bit data and potentially several START conditions, and finally a STOP condition to a target with timing requirements t_{LOW}, t_{HIGH}, $t_{HD;STA}$, $t_{SU;STA}$, $t_{SU;STO}$, and $t_{SU;DAT}$ while maintaining a bus speed of 10 kbit/s. Repeat this at least once while meeting timing requirement t_{BUF}. This test should be performed at a 25 MHz clock frequency and 125 MHz.	*I2C-014, I2C-016, I2C-018, I2C-019, I2C-020, I2C-021, I2C-022, I2C-023, I2C-024, I2C-025, I2C-026, I2C-027, I2C-028*	*Timing requirements in i2c_vu.timing_p*

T-027	Receiving fast data	For each of the I2C bus interfaces, verify that the controller can receive a START condition, followed by bit data and potentially several START conditions, and finally a STOP condition to a target with timing requirements t_{LOW}, t_{HIGH}, $t_{HD;STA}$, $t_{SU;STA}$, $t_{SU;STO}$, and $t_{SU;DAT}$ while maintaining a bus speed of 400 kbit/s. Repeat this at least once while meeting timing requirement t_{BUF}. This test should be performed at a 25 MHz clock frequency and 125 MHz.	*I2C-014, I2C-016, I2C-018, I2C-019, I2C-020, I2C-021, I2C-022, I2C-023, I2C-024, I2C-025, I2C-026, I2C-027, I2C-028*	*Timing requirements in i2c_vu.timing_p*

4.3 I2C Communication Protocol (Read/Write)

4.3.1 Communication Header

ID	Title	Description	Design Requirements	Coverage Link
T-028	Transfer start	If ControlReg.GO and BusEnReg.Bus[i] are set to 1, then the START condition should be sent on bus i for one bus clock cycle.	I2C-018, I2C-030, I2C-031, I2C-044, I2C-046	*i2c_vu.perform_read(120), i2c_vu.perform_write(191)*
T-029	Slave address	After the START condition was sent, the I2C controller should send the content of the 7 bit wide ControlReg.TargetAddress starting with the MSB on the selected bus. Test this with values 0x00, 0x01, 0x2A, and 0x7F for ControlReg.TargetAddress.	I2C-030, I2C-031, I2C-032, I2C-034, I2C-037, I2C-046	*tb_i2c_interfaces_read, tb_i2c_interfaces_write, tb_i2c_read, tb_i2c_write*
T-030	Header end	After the slave address was sent, the value on the selected bus should be set to 0 for one bus clock cycle.	I2C-030, I2C-031, I2C-046	*i2c_vu.perform_read(130), i2c_perform_write(201)*
T-031	Header ack error	If the target does not send an Ack after the selected bus was set to 0 after the slave address was sent, then ControlReg.GO should be reset to 0, StatusReg.AckError should be set to 1 and the SCL/SDA should be reset to high-Z on the selected bus until the next transfer.	I2C-017, I2C-030, I2C-031, I2C-047, I2C-048, I2C-049, I2C-046	*tb_i2c_read.stimuli_p(79), tb_i2c_write.stimuli_p(66)*
T-032	Register address	After the first Ack target was received on the selected bus, the I2C controller should send the content of ControlReg.RegisterAddress starting with the MSB on the selected bus. Test this with values 0x00, 0x01, 0xAA, 0xFF for ControlReg.RegisterAddress.	I2C-030, I2C-031, I2C-034, I2C-038, I2C-046	*tb_i2c_interfaces_read, tb_i2c_interfaces_write*

T-033	Register address ack error	If no Ack is received on the selected bus after the register address was sent, then ControlReg.GO should be reset to 0, StatusReg.AckError should be set to 1 and the SCL/SDA should be reset to high-Z on the selected bus until the next transfer.	I2C-017, I2C-030, I2C-031, I2C-046, I2C-047, I2C-048, I2C-049	*tb_i2c_read.stimuli_p(95), tb_i2c_write.stimuli_p(80)*

4.3.2 Write

ID	Title	Description	Design Requirements	Coverage Link
T-034	Data write	After the second Ack was received on the selected bus, if ControlReg.RWN is 0, then the I2C controller should send the 8 bit wide content of DataReg[0].Byte[0] starting with the MSB on the selected bus.Test this with values 0x00, 0x01, 0xAA, 0xFF for DataReg[0].Byte[0].	I2C-030, I2C-033, I2C-034, I2C-041, I2C-046	*tb_i2c_interfaces_write, i2c_vu.performe_read(76)*
T-035	Data ack	If no Ack is received on the selected bus after a data byte was sent, then ControlReg.GO should be reset to 0, StatusReg.AckError should be set to 1 and the SCL/SDA should be reset to high-Z on the selected bus until the next transfer.	I2C-017, I2C-030, I2C-046, I2C-047, I2C-048, I2C-049	*tb_i2c_write.stimuli_p(94)*
T-036	Data write multiple	If ControlReg.RWN is 0 and i-1 data bytes were sent on the selected bus in this transfer, where i-1 < ControlReg.Length <= 63, then the I2C controller should send the 8 bit wide content of DataReg[i/4].Byte[i mod 4] starting with the MSB on the selected bus. Perform this test with unique values to validate the correct byte order.	I2C-030, I2C-033, I2C-034, I2C-039, I2C-042, I2C-046	*i2c_vu.performe_read(76)*

T-037	Data length	If ControlReg.RWN is 0 and i data bytes and their Acks were sent on the selected bus in this transfer, where i = ControlReg.Length <= 63, then the I2C controller should send the STOP condition.	I2C-030, I2C-039, I2C-046	*i2c_vu.performe_read(171)*

4.3.3 Read

ID	Title	Description	Design Requirements	Coverage Link
T-038	Read Start	After the second Ack was received and if ControlReg.RWN is 1, then the START condition should be sent on the selected bus for one clock cycle.	I2C-031, I2C-040, I2C-046	*i2c_vu.perform_write(226)*
T-039	Read Slave Address	After the second START condition was sent, the I2C controller should send the content of the 7 bit wide ControlReg.TargetAddress starting with the MSB on the selected bus. Test this with values 0x00, 0x01, 0x2A, and 0x7F for ControlReg.TargetAddress.	I2C-031, I2C-032, I2C-034, I2C-037, I2C-040, I2C-046	*tb_i2c_write, i2c_vu.performe_write(196)*
T-040	Read header End	After the slave address was sent a second time, the value on the selected bus should be set to 1 for one bus clock cycle.	I2C-031, I2C-040, I2C-046	*i2c_vu.perform_write(236)*
T-041	Read header ack	If the target does not send an Ack after the selected bus was set to 1 after the slave address was sent the second time, then ControlReg.GO should be reset to 0, StatusReg.AckError should be set to 1 and the SCL/SDA should be reset to high-Z on the selected bus until the next transfer.	I2C-017, I2C-031, I2C-047, I2C-048, I2C-049, I2C-046	*i2c_write.stimuli_p(94)*

T-042	Read data byte	After an Ack from the target after setting the selected bus to 1 for one clock cycle, the received 8-bit wide data word should be in DataReg[0].Byte[0].	I2C-031, I2C-033, I2C-035, I2C-043	*tb_i2c_write*
T-043	Read data byte end	If one data byte was received and ControlReg.Length is 0, the selected bus is set to high-Z for one bus clock cycle and then the STOP condition is sent.	I2C-031, I2C-039, I2C-018	*i2c_vu.perform_write(255), i2c_vu.perform_write(262)*
T-044	Read data multiple ack	If i-1 data bytes have been received on the selected bus and i-1 < ControlReg.Length <= 63, then an Ack is sent on the selected bus.	I2C-031, I2C-035, I2C-039, I2C-046	*i2c_vu.perform_write(251), i2c_vu.perform_write(254)*
T-045	Read data multiple	If i-1 data bytes have been received on the selected bus, i-1 < ControlReg.Length <= 63 and an Ack was sent, then the received 8-bit wide data word should be in DataReg[i/4].Byte[i mod 4]. Perform this test with unique values to validate the correct byte order.	I2C-031, I2C-033, I2C-035, I2C-039, I2C-043, I2C-046	*tb_i2c_write, i2c_vu.perform_write(249)*
T-046	Read data end	If i data bytes have been received on the selected bus, where i = ControlReg.Length <= 63, the selected bus is set to high-Z for one bus clock cycle and then the STOP condition is sent.	I2C-031, I2C-035, I2C-039, I2C-018	*i2c_vu.perform_write(262)*

4.3.4 End of Transfer

ID	Title	Description	Design Requirements	Coverage Link

T-047	Transfer finish	After the STOP condition was sent, ControlReg.GO should be reset to 0.	I2C-018, I2C-045	*WaitForFlag in tb_i2c_interfaces_read, tb_i2c_interfaces_write, tb_i2c_read, tb_i2c_write*

4.4 Interrupt

ID	Title	Description	Design Requirements	Coverage Link
T-048	Interrupt	If ControlReg.GO is reset to 0 while ControlReg.IE is 1, then interrupt output should be set to 1, independent of the value of StatusReg.AckError.	I2C-029, I2C-050	*tb_i2c_interrupt.stimuli_p(35) and tb_i2c_interrupt.stimuli_p(49)*
T-049	Interrupt status flag	If ControlReg.GO is reset to 0, then StatusReg.IS should be set to 1, independent of the value of StatusReg.AckError.	I2C-029, I2C-051	*tb_i2c_interrupt.stimuli_p(37) and tb_i2c_interrupt.stimuli_p(51)*
T-050	Clearing interrupt	If StatusReg.IS is reset to 0, then interrupt output should be reset to 0.	I2C-029, I2C-052	*tb_i2c_interrupt.stimuli_p(41) and tb_i2c_interrupt.stimuli_p(55)*