



ECE 560: Assertion Based Verification

PROJECT REPORT

**ASSERTION BASED
VERIFICATION OF 8237-A
DMA CONTROLLER**

AUTHORS:

Pratik Avinash Narkhede

Minoti Sanjay Karkhanis



CONTENTS

01

OBJECTIVE

This section describes the goal of the project.

02

DESIGN OF 8237-A DMA CONTROLLER

This section describes the design implementation of the DMA Controller. It describes in brief about the modules implemented in this design.

03

VERIFICATION ENVIRONMENT

This section explains the basic verification environment.

04

FULL FPV PLAN

This section describes about the verification plan of the DMA Controller

05

HOW TO RUN THE CODE?

This section tells how to run the assertions.

06

TECHNIQUES USED

This section describes about how the assertion based verification of DMA Controller is done.

07

RESULTS

This section describes our findings in the verification

08

CHALLENGES FACED

This section describes the challenges faced due to bugs in the design and the changes which were required in the design.



CONTENTS

09

NEXT STEPS

This section describes the next steps and the future scope of this project.

10

REFERENCES

The references used in this project are mentioned in this section



OBJECTIVE

- The goal of this project is to use Formal Verification skills and verify if the design is according to the specifications using Mentor Graphics Questa PropCheck.
- The design consists of three modules- The Datapath logic, the Timing and Control Logic and the Priority Logic.
- The first step was to make a FPV Verification Plan.
- This plan would consist of the initial covers, blackboxing the priority logic module and then go for writing complex properties.
- We wrote covers to cover all the legal states in the FSM.
- We wrote covers to check if individual channels are working.
- Also, wrote covers for input output read/write and for memory read/write.
- Then we wrote assertions to check if the internal registers are cleared upon reset signal.
- We also wrote assertions for priority logic and for state machine.
- We added cut points for read operation and write operation.
- The report describes all the covers and assertions described above in detail.

DESIGN OF 8237-A DMA CONTROLLER

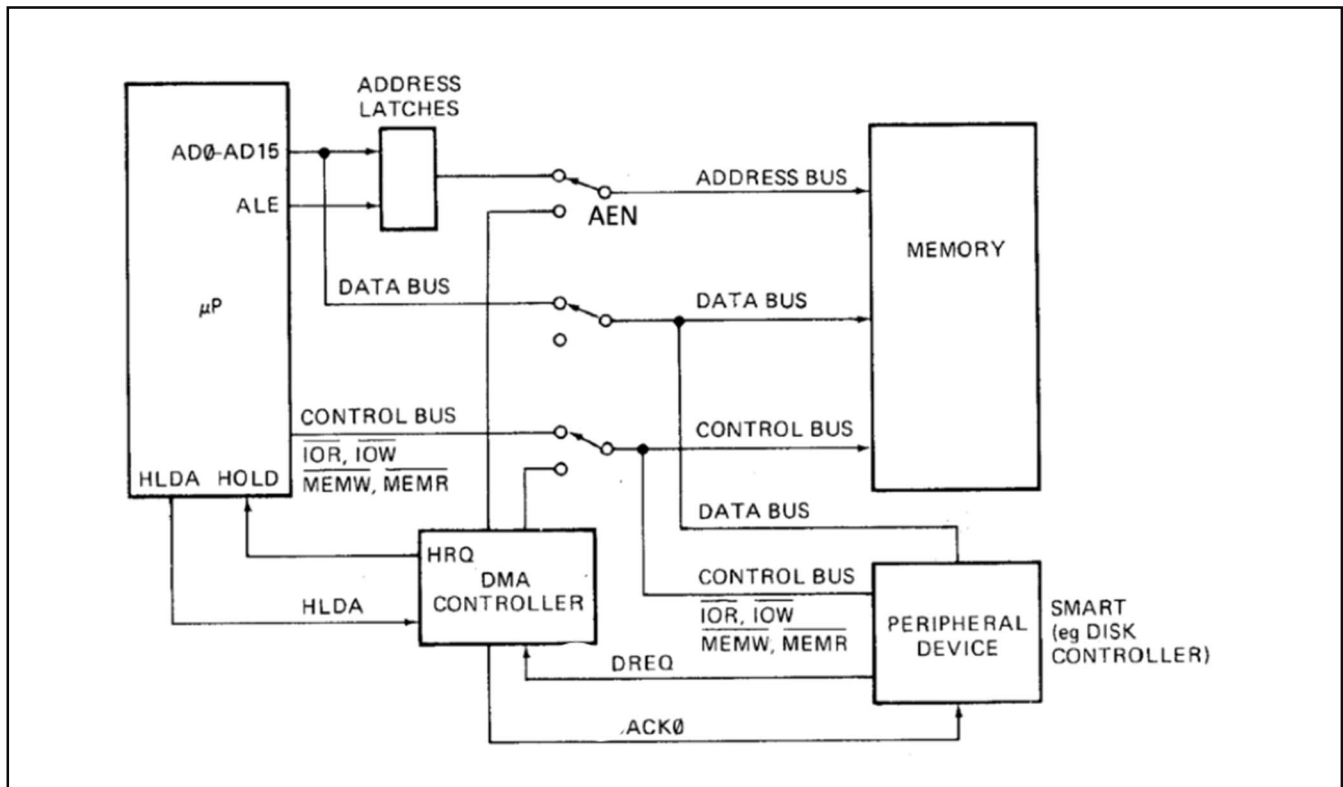


Figure 1: Block Diagram of DMA Controller interacting with Microprocessor, Memory and Peripheral devices.

- The Direct Memory Access (DMA) Controller transfers information from the I/O to memory, from memory to I/O or from memory to memory without interference of the CPU. When an I/O device requests data (DREQ) the DMA first needs acknowledgment from the CPU. The DMA sends Hold Request (HREQ) to the CPU and once the CPU gives Hold Acknowledgment (HLDA) the DMA sends Data Acknowledgment (DACK) to the I/O to indicate that the data can now be transferred.
- The design is implemented for Single Transfer Mode where the device is programmed to make one transfer only.

DESIGN OF 8237-A DMA CONTROLLER

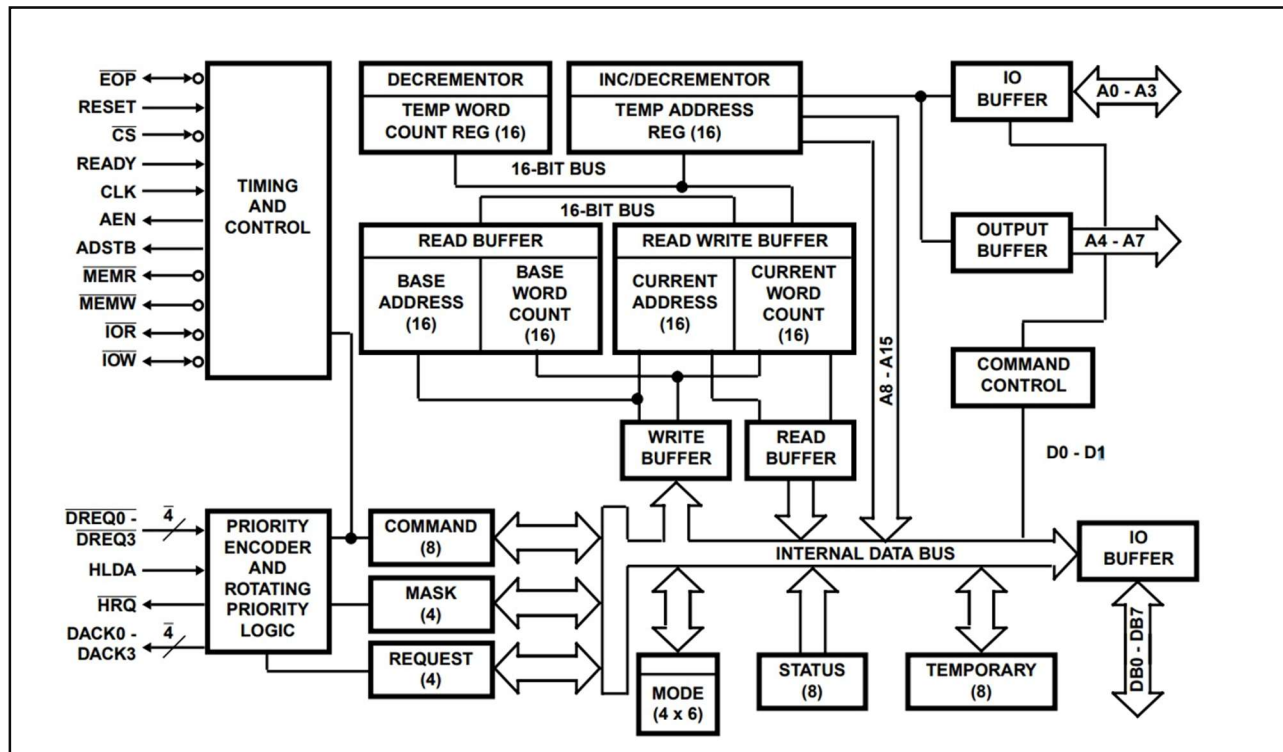


Figure 2 :Block Diagram of DMA

- The design consists of 3 main modules:
Datapath
Timing and Control
Priority Logic
- THE DATAPATH MODULE:** It consists of the following :
1) Handles the internal registers of DMA-

Command Register: This 8 bit register controls the operation of 8237. It is programmed by the microprocessor in the program condition and is cleared by Reset.

Mode Register: This 6 bit register is used to select the mode of the DMA: whether the DMA is working in Single mode, Block mode , Demand mode or Cascade mode. The microprocessor programs this register by writing bits 0 and 1.

Status Register: The Status register can be read out by the microprocessor and it gives information about the pending DMA requests.

Temporary Address Register: This register contains the value of the address in the Current Address Register. This address is incremented in the Temporary Address Register.

DESIGN OF 8237-A DMA CONTROLLER

Temporary Word Count Register: This register contains the value of the number of transfers in the Current Word Count Register. The value in the Temporary Word Count Register is decremented after every transfer.

Base Address Register: This 16 bit register stores the original value in the Current Address Register.

Current Address Register: This 16 bit register holds the value of the address used during DMA operation.

Base Word Count Register: This 16-bit register stores the original value of the Current Word Count Register.

Current Word Count Register: This 16 bit register holds the value of the number of transfers to be made during the DMA operation.

- 2) During the Idle cycle the Registers are programmed.
- 3) Increments the Address Register, decrements the Word count, and updates the status register in the Active cycle.
- 4) Makes the internal flip flop to a known state to read/write the address/word count register.

- **THE TIMING AND CONTROL MODULE:**

The Timing and Control module consists of the next state logic of the DMA Controller.

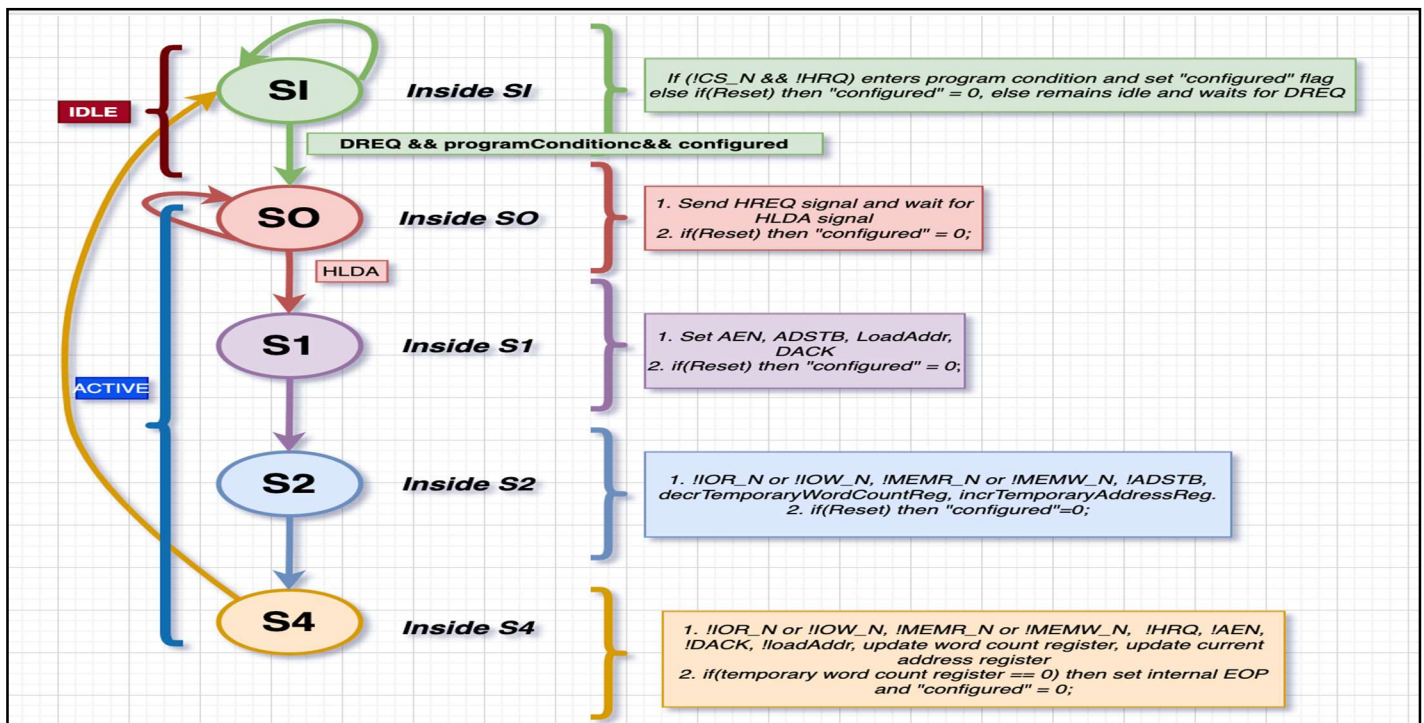


Figure 3: State transition diagram of 8237-A DMA Controller

DESIGN OF 8237-A DMA CONTROLLER

- **THE PRIORITY LOGIC MODULE:**

The DMA Controller has two types of priority logic:

Fixed Priority: Every I/O device has an unique rank. Channels are given fixed priority according to the descending order of their number.

Rotating Priority: The last channel to get acknowledgement gets the lowest priority with others rotating accordingly.

(Note:- Figure 3: State transition diagram of 8237-A DMA Controller is taken from ECE 571 Project Report)

VERIFICATION ENVIRONMENT

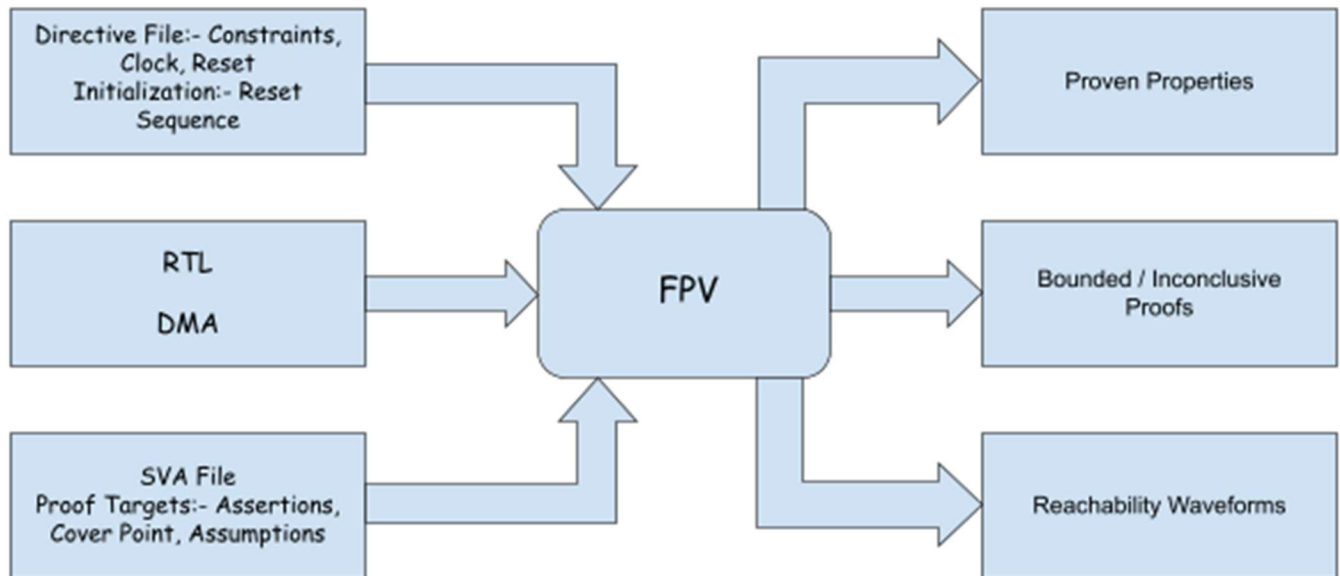


Figure 4: Block diagram of verification environment

This is a high level block diagram of the verification environment which tells us how the environment is built in order to run the assertions.

- We have directives.tcl file where we defined the clock,reset and any constraints. This is the same file where we can define cut points and blackbox.
- We have the initialization file myinit.int where we defined the reset sequence.
- Then we have the assertions files where we have wrote all the assertions, cover points and assumptions.
- We have the top design RTL code for the design of DMA.
- Then we bind the top module with the assertion file.
- We run the code using make file
- Then once the PropCheck instance is open, we check for all the properties in the Properties window.
- We can debug the failed assertions by viewing the reachability waveforms.

FULL FPV PLAN

Goal: To verify if the 8237-A DMA Controller is performing the desired operations correctly.

Properties:

Covers:

1. Cover each legal state of FSM
2. Cover to check if individual channels are working.
3. Cover for input output read or write OR memory read or write signals
4. Cover to check if the channel gets data acknowledgement (DACK) according to the fixed or rotating priority logic.

Assertions:

1. Check if the state transitions are correct.
2. Assert data request (DREQ) to one channel and check if the same channel gets data acknowledgement (DACK) according to the fixed or rotating priority logic.
3. Assert the reset signal and check if all the internal registers are cleared.
4. Assertions to check if the correct value is getting loaded and read from the internal registers.
5. All the state transition to invalid state when end of process occurs, this is safety feature.

Assumptions:

1. Assume the DMA is always active with chip select enables.
2. Assume that End of Process is never triggered in the normal run by the CPU.

Complexity Staging:

1. Blackboxing the Priority Logic block.
2. Cut point for Read operation.
3. Cut point for Write operation.

Exit Criteria: To exercise all possible reset cases, legal state transitions, legal load of registers in Program condition and update of internal registers. We exercise the priority logic according to the correct priority type.



HOW TO RUN THE CODE?

There are four types of runs to check the assertions:

- 1) *Normal Run*: The design in normal run where we check the assertions in the runtime.
For normal run execute the following command in the shell: **make run**
- 2) *Reset*: The design in reset where we check the assertions in reset mode.
For reset run execute the following command in the shell: **make reset**
- 3) *End Of Process*: The design in end of process where we check the assertions in eop mode
For end of process execute the following command in the shell: **make eop**
- 4) *Blackbox*: When we blackbox the priority logic.
For running the design with priority logic black-box, execute the following command in the shell: **make runb**

TECHNIQUES USED

- We created a reference model to capture conditions to trigger few assertions.
- We used local variables in the assertions
- We wrote generate block for exhaustive testing of fixed priority logic.
- Initially we black-boxed the priority logic module.
- We injected a bug in the datapath module to check if the loading command register fails and is captured by the assertion "loadCommandReg_a"
- Then we injected a bug in the state machine to check if state transition assertions capture any wrong state transitions.
- We also, injected bug in priority logic to check if channel 0 which is highest priority doesn't get the data acknowledgment in case of request from multiple channels.

1. BLACKBOX

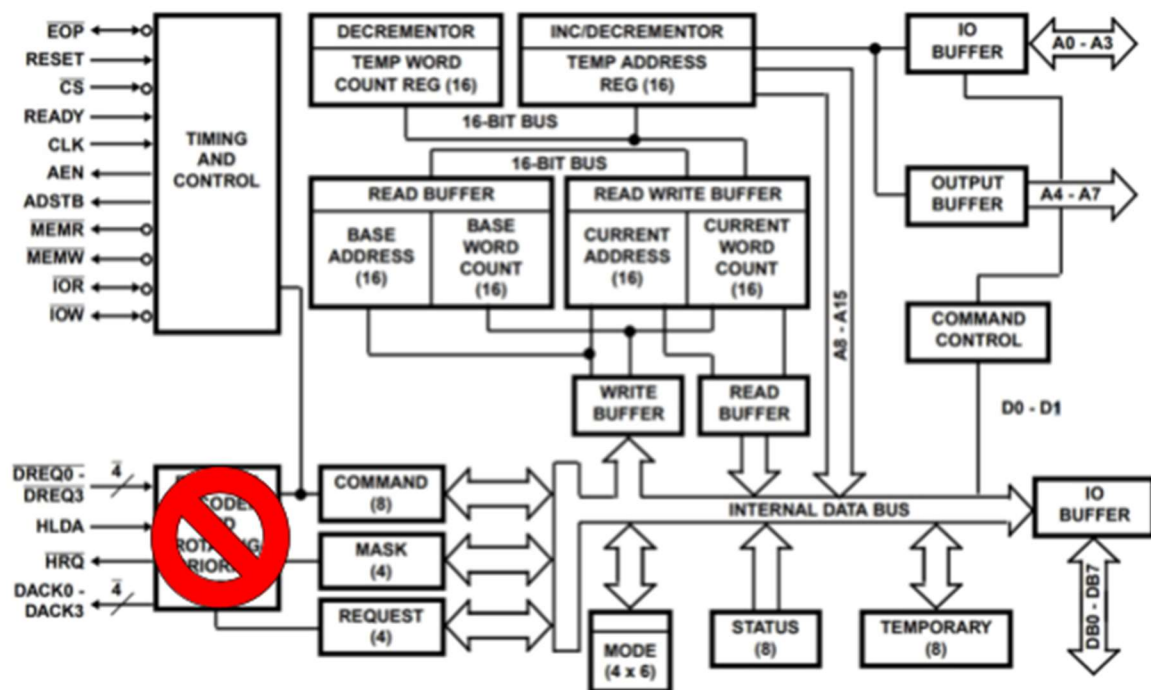


Figure 5 : Block Diagram with Blackboxed priority logic

TECHNIQUES USED

```
1  # Define clocks
2  netlist clock busIf.CLK -period 10
3
4  # Constrain rst
5  formal netlist constraint busIf.RESET 1'b0
6
7
8  netlist blackbox instance dma.pL
9
```

Figure 6: Syntax for blackbox in directivesBlackbox.tcl

2) CUT POINTS

```
1  # Define clocks
2  netlist clock busIf.CLK -period 10
3
4  # Constrain rst
5  formal netlist constraint busIf.RESET 1'b0
6
7
8  #cutpoint for ioDataBuffer
9  #netlist cutpoint dma.d.ioDataBuffer
10
11
12 #cutpoint for read
13 netlist cutpoint busIf.IOR_N
14
15 #cutpoint for write
16 #netlist cutpoint busIf.IOW_N
17
```

Figure 7: Syntax for cut point for read operation in directives.tcl

TECHNIQUES USED

```
1  # Define clocks
2  netlist clock busIf.CLK -period 10
3
4  # Constrain rst
5  formal netlist constraint busIf.RESET 1'b0
6
7
8  #cutpoint for ioDataBuffer
9  #netlist cutpoint dma.d.ioDataBuffer
10
11
12  #cutpoint for read
13  #netlist cutpoint busIf.IOR_N
14
15  #cutpoint for write
16  netlist cutpoint busIf.IOW_N
17
```

Figure 8: Syntax for cut point for write operation in directives.tcl

TECHNIQUES USED

3) PROPERTIES IN RESET MODE

Assertions to check registers and signals upon reset

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
stateTransistionOnReset_a	To check if the state transitions to SI state upon reset	Proven
commandRegZeroOnReset_a	To check if Command Register is cleared upon reset	Proven
statusRegZeroOnReset_a	To check if Status Register is cleared upon reset	Proven
modeRegZeroOnReset_a	To check if Mode Register is cleared upon reset	Proven
writeBufferZeroOnReset_a	To check if Write buffer is cleared upon reset	Proven
readBufferZeroOnReset_a	To check if Read buffer is cleared upon reset	Proven
baseAddressRegZeroOnReset_a	To check if Base Address Register is cleared upon reset	Proven
currentAddressRegZeroOnReset_a	To check if Current Address Register is cleared upon reset	Proven
baseWordCountRegZeroOnReset_a	To check if Base Word Count Register is cleared upon reset	Proven
currentWordCountRegZeroOnReset_a	To check if Current Word Count Register is cleared upon reset	Proven
tempAddressRegZeroOnReset_a	To check if Temporary Address Register is cleared upon reset	Proven
tempWordCountRegZeroOnReset_a	To check if Temporary Word Count Register is cleared upon reset	Proven
internalFFzeroOnReset_a	To check if Internal IF is cleared upon reset	Proven

TECHNIQUES USED

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
ioAddressBufferZeroOnReset_a	To check if I/O Address Buffer is cleared upon reset	Proven
outputAddressBufferZeroOnReset_a	To check if output Address Buffer is cleared upon reset	Proven
priorityOrderDefaultOnReset_a	To check if Priority Order Default is cleared upon reset	Proven
DACKisZeroOnReset_a	To check if DACK is cleared upon reset	Proven

TECHNIQUES USED

4) PROPERTIES IN NORMAL RUN MODE

ASSUMPTIONS

ASSUMPTION	DESCRIPTION	PROVEN/NOT PROVEN
dmaIsAlwaysActive	Chip Select is always enabled	Proven
noEndOfProcess_assume	No external interrupt from CPU	Proven

COVERS

Cover to check if individual channels are working

COVER	DESCRIPTION	COVERED/UNC COVERED
DACK0isOne_c	Check if DACK for Channel 0 is asserted	Covered
DACK1isOne_c	Check if DACK for Channel 1 is asserted	Covered
DACK2isOne_c	Check if DACK for Channel 2 is asserted	Covered
DACK3isOne_c	Check if DACK for Channel 3 is asserted	Covered

TECHNIQUES USED

Cover for input output read or write OR memory read or write signals

COVER	DESCRIPTION	COVERED/UNC COVERED
ioRead_c	Check if the I/O Read IOR_N signal is being asserted	Covered
ioWrite_c	Check if the I/O Read IOW_N signal is being asserted	Covered
memoryRead_c	Check if the I/O Read MEMR_N signal is being asserted	Covered
memoryWrite_c	Check if the I/O Read MEMW_N signal is being asserted	Covered

Cover for Signals

COVER	DESCRIPTION	COVERED/UNC COVERED
AENactive_c	Cover to check if Address Enable is active	Covered
ADSTBactive_c	Cover to check if Address Strobe is active	Covered
HRQactive_c	Cover to check if Hold Request is active	Covered

TECHNIQUES USED

Cover for checking all the legal states of FSM

COVER	DESCRIPTION	COVERED/UNCOVERED
stateSI_c	To check if the DMA enters SI state	Covered
stateSO_c	To check if the DMA enters SO state	Covered
stateS1_c	To check if the DMA enters S1 state	Covered
stateS2_c	To check if the DMA enters S2 state	Covered
stateS4_c	To check if the DMA enters S4 state	Covered
stateTransitions_c	To check if the DMA transitions from SI state to SO state to S1 state to S2 state to S4 state and back to SI state	Covered

Covers inside generate block for Fixed Priority logic (exhaustive testing)

COVER	DESCRIPTION	COVERED/UNCOVERED
DACK0001forDREQ_c	In the case where channel 0 DREQ is active then DACK should be 0001	Covered
DACK0010forDREQ_c	When channel 0 DREQ is not active and channel 1 DREQ is active DACK should be 0010	Covered
DACK0100forDREQ_c	When channel 0,1 DREQ is not active and channel 2 DREQ is active DACK should be 0100	Covered
DACK1000forDREQ_c	When only channel 3 DREQ is active DACK should be 0010	Covered
DACK0000forDREQ_c	When there's no request from any channel DACK should be 0000	Proven

TECHNIQUES USED

Cover for Rotating priority logic

COVER	DESCRIPTION	COVERED/UNCOVERED
rotatingPriority_c	DREQ 1111 is asserted everytime. Check if each channel gets Data Acknowledgment one after the other.	Covered

ASSERTIONS

Assertions for state machine

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
stateTransistionSItoSO_a	Verify if the state transitions from SI to SO	Proven
stateTransistionSOtoS1_a	Verify if the state transitions from SO to S1	Proven
stateTransistionS1toS2_a	Verify if the state transitions from S1 to S2	Proven
stateTransistionS2toS4_a	Verify if the state transitions from S2 to S4	Proven
stateTransistionS4toSI_a	Verify if the state transitions from S4 to SI	Proven

TECHNIQUES USED

Assertions inside generate block for fixed priority logic (exhaustive testing):

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
DACK0001forDREQfixedPriority_a	In the case where channel 0 DREQ is active then DACK should be 0001	Proven
DACK0010forDREQfixedPriority_a	When channel 0 DREQ is not active and channel 1 DREQ is active DACK should be 0010	Proven
DACK0100forDREQfixedPriority_a	When channel 0,1 DREQ is not active and channel 2 DREQ is active DACK should be 0100	Proven
DACK1000forDREQfixedPriority_a	When only channel 3 DREQ is is active DACK should be 0010	Proven
DACK0000forDREQfixedPriority_a	When there's no request from any channel DACK should be 0000	Proven

Assertions to load registers

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
loadIoDataBufferFromDB_a	Check if I/O data buffer is loaded with the databus value	Proven
loadCommandReg_a	Check if Command Register is loaded with value of i/o data buffer	Proven
loadWriteBuffer_a	Check if write buffer is loaded with value of i/o data buffer	Vacuously passed
loadBaseUpperAddress_a	Check if upper bits of base address register is loaded with the value of write buffer	Proven
loadBaseLowerAddress_a	Check if lower bits of base address register is loaded with the value of write buffer	Proven

TECHNIQUES USED

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
loadCurrentUpperAddress_a	Check if upper bits of current address register is loaded with the value of write buffer	Proven
loadCurrentLowerAddress_a	Check if lower bits of current address register is loaded with the value of write buffer	Proven
loadBaseUpperWordCount_a	Check if upper bits of base word count register is loaded with the value of write buffer	Proven
loadBaseLowerWordCount_a	Check if lower bits of base word count register is loaded with the value of write buffer	Proven
loadCurrentUpperWordCount_a	Check if upper bits of current word count register is loaded with the value of write buffer	Proven
loadCurrentLowerWordCount_a	Check if lower bits of current word count register is loaded with the value of write buffer	Proven

Assertions to read registers

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
readCurrentUpperAddress_a	Check if upper bits of current address register are captured in read buffer	Proven
readCurrentLowerAddress_a	Check if lower bits of current address register are captured in read buffer	Proven
readCurrentUpperWordCount_a	Check if upper bits of current word count register are captured in read buffer	Fired
readCurrentLowerWordCount_a	Check if lower bits of current word count register are captured in read buffer	Fired
readStatusReg_a	Check if status register captured in read buffer i/o data buffer	Fired

TECHNIQUES USED

Assertions for Timing Control and Datapath

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
internalFFzero_a	Check if the value of internal flipflop is 0	Proven
internalFFone_a	Check if the value of internal flipflop is 1	Proven
baseAddressShouldNotChange_a	Check is base address register is not changed	Fired
baseWordCountShouldNotChange_a	Check if base word count register is not changed	Fired
singleRegConfigInProgramMode_a	Check if only 1 register is active at a time for configuration in program mode	Proven
addressEnable_a	Check if address is enabled after hold acknowledge	Proven
addressEnableIsActiveForTwoCycles_a	Check if address enable is active for two cycles	Proven
addressStrobeActive_a	Check is address strobe is active after hold acknowledgement	Proven
addressStrobeActiveforOneCycle_a	Check if address strobe is active for one cycle	Proven
addressBusValid_a	Check if address bus is valid after address strobe	Proven
dataBusValid_a	Check if databus is valid after address strobe and address enable	Proven
noReadWriteAtSameTime_a	Check if there's no read and write at same time	Fired
addressValidOnReadWrite_a	Check if address is valid during read or write operation	Proven
dataValidOnReadWrite_a	Check if databus is valid during write operation	Proven
validReadWriteSignalsOnHLDA_a	Check if read and write signals are known after hold acknowledgement	Proven
validAddressBusOnHLDA_a	Check if address bus is valid after hold acknowledgement	Proven

TECHNIQUES USED

5) PROPERTIES IN EOP MODE

Assertions for state transition on End of Process

*This is **safety feature** where all the state should transition to Invalid State when the end of process occurs.*

ASSERTION	DESCRIPTION	PROVEN/NOT PROVEN
stateTransistionSItoSIonEOP_a	Check if state remains invalid on end of process.	Proven
stateTransistionSOtoSIonEOP_a	Check if state transitions from SO to invalid on end of process	Proven
stateTransistionS1toSIonEOP_a	Check if state transitions from S1 to invalid on end of process	Proven
stateTransistionS2toSIonEOP_a	Check if state transitions from S2 to invalid on end of process	Proven
stateTransistionS4toSIonEOP_a	Check if state transitions from S4 to invalid on end of process	Proven

RESULTS

Results for Covers:

<input type="checkbox"/>			check1.DACK0isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.DACK1isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.DACK2isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.DACK3isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.ioRead_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.ioWrite_c	★ 7	11 @ buslf.CLK	0s
<input type="checkbox"/>			check1.memoryRead_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.memoryWrite_c	★ 7	11 @ buslf.CLK	0s
<input type="checkbox"/>			check1.AENactive_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.ADSTBactive_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.HRQactive_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateSI_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateSO_c	★ 7	2 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateS1_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateS2_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateS4_c	★ 7	5 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateTransitions_a	★ 7	15 @ buslf.CLK	0s

<input type="checkbox"/>			check1.g1[0].DACK0000.DACK0000forDREQ_c	★ 10	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[1].DACK0001.DACK0001forDREQ_c	★ 10	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[2].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[3].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[4].DACK0100.DACK0100forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[5].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[6].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[7].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[8].DACK1000.DACK1000forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[9].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[10].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[11].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[12].DACK0100.DACK0100forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[13].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[14].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[15].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.rotatingPriority_c	★ 7	10 @ buslf.CLK	0s

RESULTS

Results for Covers during Run

<input type="checkbox"/>			check1.g1[0].DACK0000.DACK0000forDREQ_c	★ 10	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[1].DACK0001.DACK0001forDREQ_c	★ 10	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[2].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[3].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[4].DACK0100.DACK0100forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[5].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[6].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[7].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[8].DACK1000.DACK1000forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[9].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[10].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[11].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[12].DACK0100.DACK0100forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[13].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[14].DACK0010.DACK0010forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.g1[15].DACK0001.DACK0001forDREQ_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.rotatingPriority_c	★ 7	10 @ buslf.CLK	0s
<input type="checkbox"/>			check1.DACK0isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.DACK1isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.DACK2isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.DACK3isOne_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.ioRead_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.ioWrite_c	★ 7	11 @ buslf.CLK	0s
<input type="checkbox"/>			check1.memoryRead_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.memoryWrite_c	★ 7	11 @ buslf.CLK	0s
<input type="checkbox"/>			check1.AENactive_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.ADSTBactive_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.HRQactive_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateSI_c	★ 7	6 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateSO_c	★ 7	2 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateS1_c	★ 7	3 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateS2_c	★ 7	4 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateS4_c	★ 7	5 @ buslf.CLK	0s
<input type="checkbox"/>			check1.stateTransitions_a	★ 7	15 @ buslf.CLK	0s

RESULTS

Results for State Transitions during the End of Process (EOP)




<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Name	Health	Radius	Time
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check3.stateTransistionSltoSlonEOP_a	★ 10		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check3.stateTransistionSOtoSlonEOP_a	★ 10		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check3.stateTransistionS1toSlonEOP_a	★ 10		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check3.stateTransistionS2toSlonEOP_a	★ 10		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check3.stateTransistionS4toSlonEOP_a	★ 10		0s

Results for assertions on reset

















<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Name	Health	Radius	Time
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.stateTransistionOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.commandRegZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.statusRegZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.modeRegZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.writeBufferZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.readBufferZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.baseAddressRegZeroOnReset_a	★ 0		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.currentAddressRegZeroOnReset_a	★ 0		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.baseWordCountRegZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.currentWordCountRegZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.tempAddressRegZeroOnReset_a	★ 7		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.tempWordCountRegZeroOnReset_a	★ 0		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.internalFFzeroOnReset_a	★ 0		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.ioAddressBufferZeroOnReset_a	★ 0		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.outputAddressBufferZeroOnReset_a	★ 0		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.priorityOrderDefaultOnReset_a	★ 0		0s
<input type="checkbox"/>	<input checked="" type="radio"/>	<input checked="" type="checkbox"/>	check2.DACKisZeroOnReset_a	★ 0		0s

RESULTS

Results for Assumptions

		check1.dmalsAlwaysActive
		check1.noEndOfProcess_assume

Failed Assertions and vacuously passed assertions

		Name	Health	Radius	Time
<input type="checkbox"/>		check1.noReadWriteAtSameTime_a	 10	1 @ buslf.CLK	0s
<input type="checkbox"/>		check1.readStatusReg_a	 7	12 @ buslf.CLK	0s
<input type="checkbox"/>		check1.baseAddressShouldNotChange_a	 7	1 @ buslf.CLK	0s
<input type="checkbox"/>		check1.baseWordCountShouldNotChange_a	 7	1 @ buslf.CLK	0s
<input type="checkbox"/>		check1.readCurrentUpperWordCount_a	 7	5 @ buslf.CLK	0s
<input type="checkbox"/>		check1.readCurrentLowerWordCount_a	 7	5 @ buslf.CLK	0s
<input type="checkbox"/>		check1.loadWriteBuffer_a	 7		0s

Possible reason for assertion getting fired or vacuous pass:-

- Design has multiply-driven error due to which read status register and read current word count was fired.
- Load write buffer was vacuously passed because the condition to load the write buffer was never satisfied.

RESULTS

Property Window Summary for Reset Mode

9	Property Summary		
10	-----		
11	Total	Directives :17	
12	Assert	Directives :17	
13	Assume	Directives :0	
14	Cover	Directives :0	
15			
16			
17	SVA Summary		
18	-----		
19	SVA Directive Type	Directives	Checkers
20	-----		
21	Assert	17	17
22	Assume	0	0
23	Cover	0	0
24	-----		

Property Window Summary for Run Mode

9	Property Summary		
10	-----		
11	Total	Directives :89	
12	Assert	Directives :53	
13	Assume	Directives :2	
14	Cover	Directives :34	
15			
16			
17	SVA Summary		
18	-----		
19	SVA Directive Type	Directives	Checkers
20	-----		
21	Assert	53	53
22	Assume	2	2
23	Cover	34	34
24	-----		

RESULTS

Property Window Summary for EOP Mode

9	Property Summary		
10	-----		
11	Total	Directives :5	
12	Assert	Directives :5	
13	Assume	Directives :0	
14	Cover	Directives :0	
15			
16			
17	SVA Summary		
18	-----		
19	SVA Directive Type	Directives	Checkers
20	-----		
21	Assert	5	5
22	Assume	0	0
23	Cover	0	0
24	-----		

EFFORT REQUIRED (Approximate no. of hours needed to complete the project)

We required approximately 65 hours to complete the project.



CHALLENGES FACED

- There were inferred latches in the design which we fixed by modifying the design.
- There was a combinational loop in the priority logic. The combinational loop is something where in the `always_comb` block we get some output and we use the same output as the input in the block. we fixed this problem by adding a d-flipflop
- We had multiply-driven variables in the design. This was due to the same variable being used in multiple `always_ff` blocks. We removed a few multiply-driven variable problems.
- The state machine had a problem in the output of idle state where the program condition's internal variable was wrongly updated.
- `tempAddressReg` was failing on reset because in the design it was initialized 2 times. We fixed this problem in the datapath module.
- We tried to write liveness property using `s_eventually` in few assertions. For instance, the state machine can stay in idle state for long time till the CPU handles over the control of bus. We tried to write this state transition using `s_eventually` but the assertion always failed.

NEXT STEPS

- We need to remodel few parts of the code to remove multiply driven problem from the code.
- The readCurrentUpperWordCount_a, readCurrentLowerWordCount_a ,readStatusReg_a , baseAddressShouldNotChange_a , baseWordCountShouldNotChange_a and noReadWriteAtSameTime_a were fired due the multiply driven problem and we would fix this problem in future.
- The condition for loadWriteBuffer_a is not being satisfied and hence it is vacuously passed. We will check the condition in the design and would fix the problem.
- In the design, the address, data and channels are parameterized so we will increase the parameter value to increase the complexity of the project.
- Implement the block transfer and cascade transfer mode in the design and further write assertions to verify these modes and at the same time make sure that the current assertions still pass on the updated design.

REFERENCES

- 8237A High Performance Programmable DMA Controller Datasheet by Intel.
- The design of 8237-A DMA Controller was done during the course ECE 571 in Spring 2021 under the guidance of Prof. Mark Faust by:
Pratik Avinash Narkhede
Tanmay Nitin Patil
Abdul Hasan Imroze Mohammed
- ECE 560 Assertion Based Verification Slides by Prof. Tom Schubert.
- “Formal verification: an essential toolkit for modern vlsi design” , by Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar.