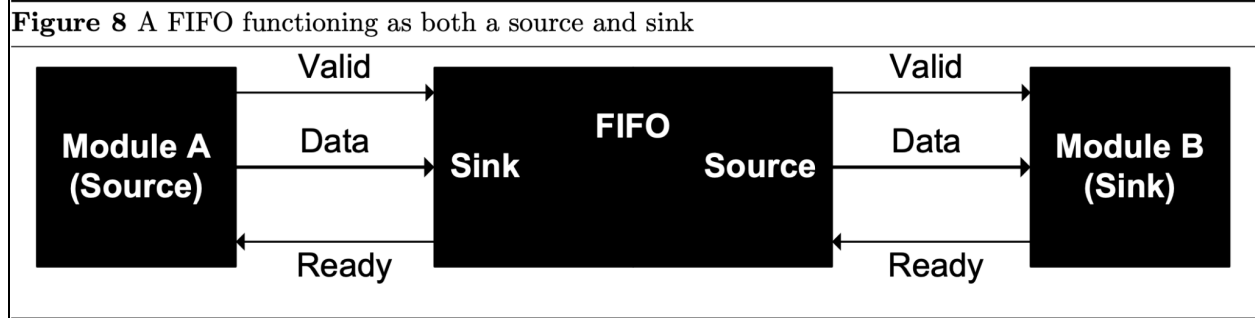


# [EECS 151/251A] Assertion-Based Verification Lab Specification

**Overview:** This lab will help you grow your assertion-based verification knowledge and develop skills like FSM comprehension and assertion writing. Assertion-based verification is an increasingly important step in hardware design, and industry is paying attention. We hope that this assignment will equip you with skills needed to pursue a successful career in hardware verification. In this lab, you will write SystemVerilog Assertions to functionally verify the FIFO shown in Figure 8 of the pre-lab document (copied below for your convenience).



**Pre-lab:** Prior to coming into your assigned lab section, you must read through and understand [this pre-lab document](#), which details a handshake-based, ready-valid interface protocol for transmitting data from one component of a design (source) to another (sink).

**Background:** The standard approach to verification is to write assertions on the signals being driven by the module of interest (in this case, the outgoing signals of the FIFO: Ready -> Module A (Source), Valid + Data -> Module B (Sink), as the behavior of these signals with respect to design expectations lies under the jurisdiction of the FIFO itself. Conversely, from the FIFO's perspective, it will need to make assumptions about its incoming signals, in particular that they are also adhering to the same handshake-based, ready-valid interface protocol.

We will model the FIFO buffer as a Finite State Machine (FSM), similar to what you have seen in previous lectures, discussions, and labs. This will help us write assertions. Below we provide the description of the FSM.

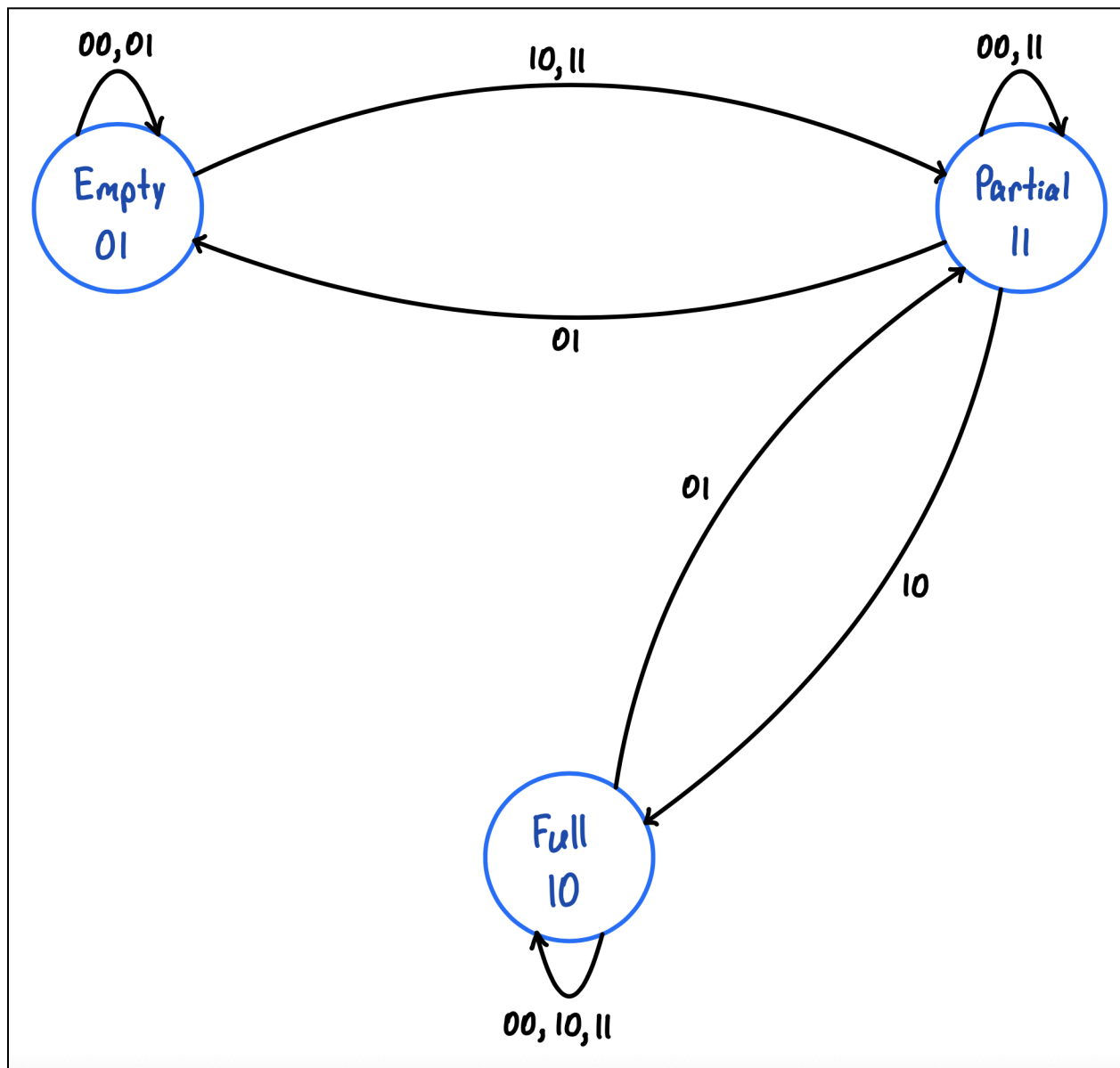
**FIFO Design:** We will provide a FSM implementation of the FIFO depicted above under the following three simplifying assumptions:

1. The depth of the FIFO is 2 (i.e., it can store at most 2 pieces of data at any point in time).
  2. Data written on a given cycle cannot also be read on that same cycle.
  3. Only one piece of data can be read from the FIFO on a given cycle.
- Inputs: 2'bXX
    - Write
    - Read
      - Take the form of {Write, Read}
  - Outputs: 2'bXX
    - Valid
    - Ready

# [EECS 151/251A] Assertion-Based Verification Lab Specification

- Take the form of {Valid, Ready}
- States: 2'bXX
  - Empty = 2'b01
  - Partial = 2'b11
  - Full = 2'b10

**FIFO FSM State Transition Diagram:**



**FIFO FSM Encoded State Transition Table:**

Current State	Input	Next State
---------------	-------	------------

# [EECS 151/251A] Assertion-Based Verification Lab Specification

{0,1} (Empty)	{0,0} (!write & !read)	{0,1} (Empty)
{0,1} (Empty)	{0,1} (!write & read)	
{0,1} (Empty)	{1,0} (write & !read)	
{0,1} (Empty)	{1,1} (write & read)	
{1,1} (Partial)	{0,0} (!write & !read)	{1,1} (Partial)
{1,1} (Partial)	{0,1} (!write & read)	
{1,1} (Partial)	{1,0} (write & !read)	
{1,1} (Partial)	{1,1} (write & read)	
{1,0} (Full)	{0,0} (!write & !read)	{1,0} (Full)
{1,0} (Full)	{0,1} (!write & read)	
{1,0} (Full)	{1,0} (write & !read)	
{1,0} (Full)	{1,1} (write & read)	

Current State	Output
{0,1} (Empty)	{0,1} (!valid & ready)
{1,1} (Partial)	
{1,0} (Full)	

In other words, the current state of the design is itself the output, given this convenient 1:1 mapping of the bit-level representations of the current state and output.

**Instructions:** Given a Verilog implementation of the above FIFO FSM, write a comprehensive set of assertions that verify the state transitions, output, and interface protocol as outlined in the pre-lab document. A couple sample assertions are filled in below to get you started.

## FIFO State Transition Assertion (non-overlapping implication):

- Empty state transitions
  - `assert property((@posedge clk) disable iff (rst) (state==2'b01 && write && !read) |=> (next_state==2'b11));`

## Output Assertion (overlapping implication):

- `assert property((@posedge clk) disable iff (rst) (state==2'b01) |-> (!valid && ready));`

## **Solution Assertions:**

## FIFO State Transition Assertions (non-overlapping implications):

- Empty state transitions

## [EECS 151/251A] Assertion-Based Verification Lab Specification

- assert property((@posedge clk) disable iff (rst) (state==2'b01 && write && !read) | => (next\_state==2'b11));
- assert property((@posedge clk) disable iff (rst) (state==2'b01 && !write) | => (next\_state==2'b01));
- Partial state transitions:
  - assert property((@posedge clk) disable iff (rst) (state==2'b11 && write && !read) | => (next\_state==2'b10));
  - assert property((@posedge clk) disable iff (rst) (state==2'b11 && !write && read) | => (next\_state==2'b01));
  - assert property((@posedge clk) disable iff (rst) (state==2'b11 && ((write && read) || (!write && !read))) | => (next\_state==2'b11));
- Full state transitions:
  - assert property((@posedge clk) disable iff (rst) (state==2'b10 && !write && read) | => (next\_state==2'b11));
  - assert property((@posedge clk) disable iff (rst) (state==2'b10 && (!read && !write || write)) | => (next\_state==2'b10));

### Output Assertions (overlapping implications):

- assert property((@posedge clk) disable iff (rst) (state==2'b01) |-> (!valid && ready));
- assert property((@posedge clk) disable iff (rst) (state==2'b11) |-> (valid && ready));
- assert property((@posedge clk) disable iff (rst) (state==2'b10) |-> (valid && !ready));

### Interface Protocol Assertions:

- assert property((@posedge clk) disable iff (rst) !(state==2'b01 && read)); <- no read when empty
- assert property((@posedge clk) disable iff (rst) !(state==2'b10 && write)); <- no write when full