FSM Cookbook

1. Introduction

Tau models describe the timing and functional information of component interfaces. Timing information specifies the delay in placing values on output signals and the timing constraints (set-up/hold, pulse-width) on input signals of a component. Functional information, through a finite state machine (FSM), specifies when output signal values change, when input signal values are latched, and how output values are determined as a function of input values. Tau reasons over FSMs to eliminate the reporting of false timing violations. While describing timing information is straightforward, users often are unsure about what functional information must be captured in an FSM and the best approach to doing so. This document provides a strategy for specifying the functional information of component interfaces for timing analysis within Tau.

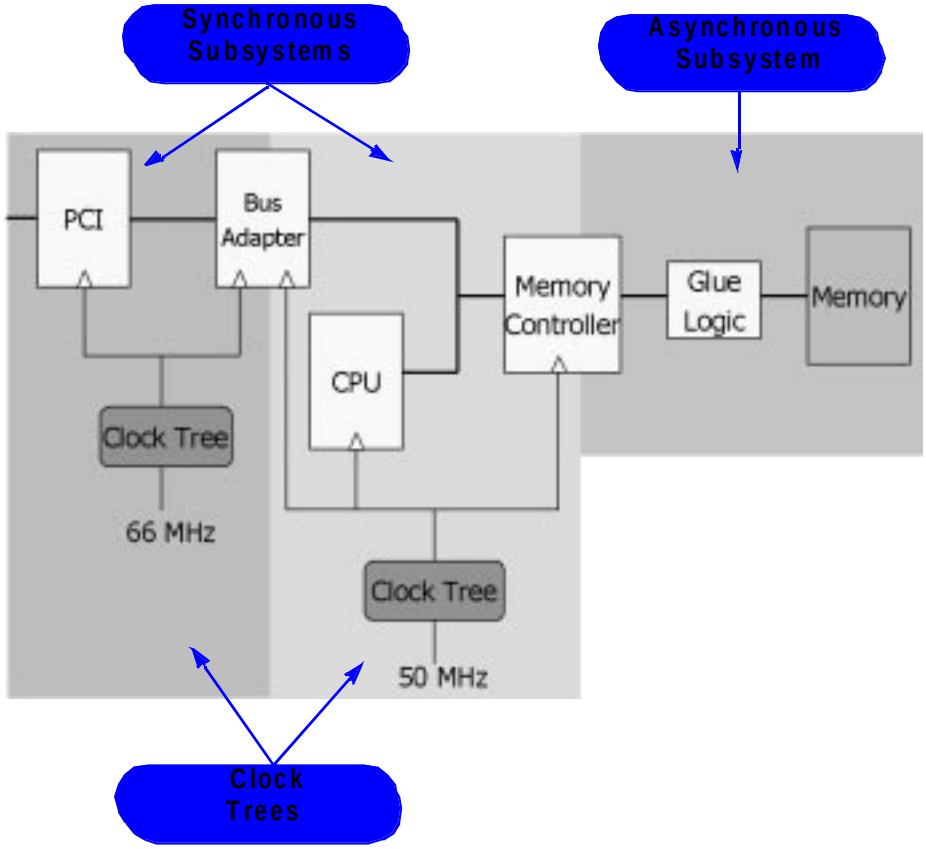
2. What Functional Information Must be Modeled

When creating a Tau model for a component it is first necessary to understand what functional information needs to be captured in the model. In general, the functional information that is useful to Tau describes the manner in which a component interfaces to the outside world. Such information typically specifies when output signal values change, when input signal values are latched, and the values driven on control signals. This information is commonly specified through a timing diagram. It is important to note that details on the internal functionality of a component are of no interest to Tau. If a timing diagram does not easily capture some functional information on a component then the information is probably not useful for timing analysis purposes and can be ignored. When debating whether to provide some functional information always trade-off the ease of providing this information with the improvement in verification accuracy. If by providing a little information you can dramatically improve verification accuracy, then do so. If, however, describing some functional behavior is tedious and the improvement in verification accuracy is minimal, then do not bother to enter such information.

Depending on where a component is instantiated on a design, it is sometimes possible to either not specify any functional information about the component, or to automatically generate default functional information from the timing specification for the component. It is important to know when this can be done because it significantly reduces the modeling burden a user encounters. With this in mind, it is useful to view board-level circuits as composed of three types of subsystems (shown in Figure1):

1. Clock trees.
2. Synchronous subsystem.
3. Asynchronous subsystem.

Figure 1: Typical Board-Level Circuit



2.1 Clock Trees

Clock trees contain components such as clock buffers, phase-locked loops, frequency dividers and oscillators. For all components in a clock tree, you do *not* need to provide functional information in a Tau model. That is, you do not to create any FSMs for such components. The central focus of clock tree analysis is to ensure that the skew and phase shift between the sink points of a clock tree are within acceptable bounds. This task can be performed without regard to the functionality of components in the clock. The only information that is required to perform clock tree analysis is the skew, or correlation in delay, among component outputs and the delay from input to output. Therefore, if a component you are creating a model for is used only in the clock tree of your design, you do not need to create or obtain any FSMs for the component.

2.2 Synchronous Subsystems

A synchronous subsystem on a board is a collection of synchronous components that are all clocked relative to the same reference clock. In turn, a synchronous component is one whose outputs change and inputs are latched relative to a clock signal. The timing information for a synchronous component is defined strictly by "clock-to-output" delays and set-up/hold times on inputs relative to a clock. A synchronous subsystem can have clocks with phase shift, skew, and frequency variations among them, but it is should be possible to fix all clock waveforms relative to a common reference clock. The timing constraints on a synchronous subsystem can be verified without the functional information of components in the subsystem, but by simply manipulating component timing information (delays and constraints) and clock descriptions (frequency, phase-shift, skew and jitter). Consequently, the functional information for components belonging to a synchronous subsystem can default to a *static* model, where the triggering edge of a clock latches all inputs and changes all outputs. You can generate a static model automatically in Tau from the timing information for a component using the **Cell->Create FSM From Timing** menu item. In summary, if a component is only used in the synchronous subsystems of a design, you do not need to create its FSMs manually.

2.3 Asynchronous Subsystems

An asynchronous subsystem on a board contains components with set-up/hold times relative to non-clock signals and multi-cycle propagation paths. An example asynchronous subsystem is the memory subsystem on a board. Such a subsystem typically contains a memory controller, glue logic and memory chips (SRAM, DRAM, Flash, FIFO, etc.). To avoid dealing with large numbers of false violations it is necessary to provide functional information that accurately describe how these components interact with the outside world. So, for components that are part of an asynchronous subsystem on a board, you *should* provide functional. The next section discusses approaches to obtain or specify this information.

Always keep in mind that you need to model functional information (FSMs) for a component only if it is part of an asynchronous subsystem. If the component belongs to the clock tree or a synchronous subsystem, then you do not need functional information, or you can generate it automatically, without adversely affecting the accuracy of reported results.

3. Approaches to Obtaining/Creating Functional Information

FSMs for Tau may be obtained or generated from the following different sources:

1. Interconnectix model libraries and model development services.
2. Timing diagrams.
3. Manually creating FSMs using the Tau Library Editor.

3.1 Models from Interconnectix

Interconnectix (ICX) provides a library of Tau models for common off-the-shelf components such as glue logic (buffers, decoders, transceivers, registers, etc.), memory chips (SRAM, DRAM, Flash, etc.), processors (Motorola, IDT, etc.) and bus interfaces (PCI, ISA, etc.). To obtain this library, which is provided at no cost, contact your local Mentor Graphics applications engineer.

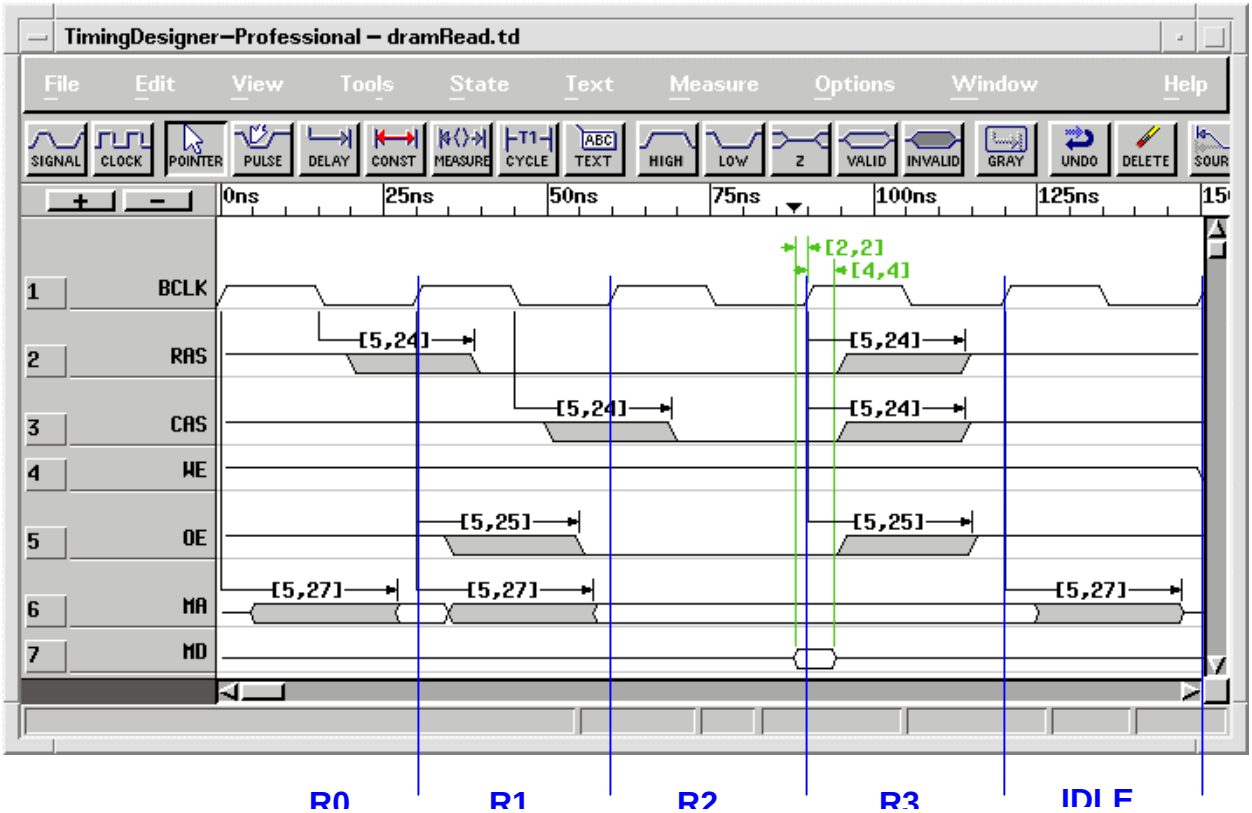
When you need to define functional information for an off-the-shelf component, it is useful to first look at the ICX model library to see if a component model already exists. If it does, you should reuse the existing model. If it does not, but a component that is very similar to one you need to model exists in the ICX library, then it is recommended that you reuse information by making the necessary modifications to the existing model.

If the components you are modeling are available off-the-shelf, but do not exist in the ICX library, then you could contract ICX to deliver models for your project. At the time this document was written the cost for these services was $3,000 per project and delivery time was 2 weeks. To obtain information on project-based modeling services offered by ICX contact your local Mentor Graphics applications engineer.

3.2 Timing Diagrams

Timing diagrams are a useful mechanism for capturing the functional information of component interfaces. Often, timing diagrams are already available as a by-product of the documentation process for a component. Figure 2 shows an example timing diagram of a DRAM read cycle. If timing diagrams are available in Chronology Corp.'s TimingDesigner format (called TD), you can import them directly into the Tau Library Editor. This produces a timing model for the component and FSMs if the behavior on the timing diagram is clocked. If the diagrams are available in a form that is compliant with the new industry standard TDML (Timing Diagram Markup Language), then you can import them into the Tau Library Editor. You do this by first reading them into TimingDesigner and then saving them as TD files.

Figure 2: Timing Diagram Example



To obtain timing diagrams for off-the-shelf components visit the Synchrony page on Chronology's web-site (www.chronology.com). You can also contact the component vendor. It is very likely that they used TimingDesigner to draw the diagrams you see in their datasheet and they may be willing to send you the TD or TDML. If you need to draw a diagram manually because it is unavailable elsewhere, then follow the steps outlined in the white paper "Using Timing Diagrams as Models for Tau".

3.3 Creating FSMs Manually

This section lists the sequence of steps that you perform when creating FSMs manually. The SRAM datasheet in Figure 3 and the timing diagram for the memory controller DRAM read access shown in Figure 2, will be used to illustrate these steps for creating asynchronous and synchronous FSMs from datasheet information.

1. Start by determining whether the FSM is synchronous or asynchronous.

An FSM is synchronous if all outputs always change and all inputs are always latched relative to a clock signal. An FSM may have at most one clock signal. Note that an asynchronous set-reset flip-flop is not synchronous because the set-reset inputs can directly change output values regardless of when the clock changes. So, while there is a state, when set and reset are de-asserted, in which the set-reset flip-flop behaves synchronously, overall the FSM exhibits asynchronous behavior. By default, an FSM is asynchronous. Use the library editor to make an FSM synchronous using the **FSM > Make Async/Sync** menu item.

The behavior for the DRAM timing diagram in Figure 2 is clocked, so the Memory Controller FSM is synchronous.

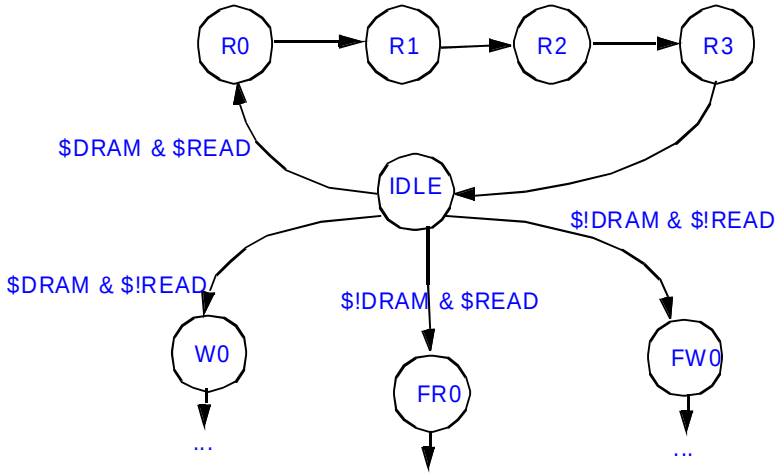
The behavior for SRAM read and write accesses in Figure 3 is not clocked, so the SRAM FSM is asynchronous.

2. Next, identify the different states on an FSM.

   Each state on a synchronous FSM may span at most one clock cycle. If an FSM has multiple states, then one of the states must be called IDLE. The IDLE state denotes the initial state of an FSM.

   The DRAM read access in Figure 2 spans five clock cycles. Therefore, the interface has five states: IDLE, R0, R1, R2 and R3. The Memory Controller Interface FSM (which includes the DRAM read cycle) is shown in Figure 3.

Figure 3: Memory Controller Interface

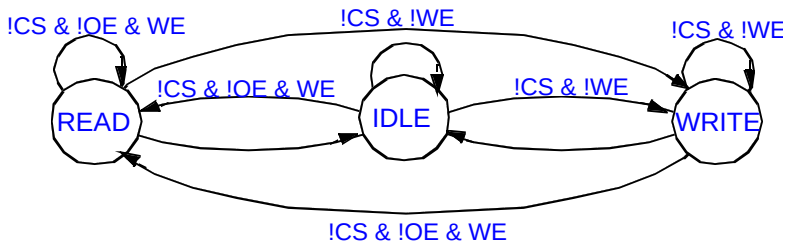


Similarly, the SRAM truth table in Figure 4 indicates that the SRAM has three states: IDLE, READ and WRITE (the “outputs disabled” and “deselected” states are combined into a single IDLE state). The SRAM FSM is shown in Figure 5.

Figure 4: SRAM Truth Table

CS	OE	WE	I/O	Function
L	L	H	$DATA_{OUT}$	Read Data
L	X	L	$DATA_{IN}$	Write Data
L	H	H	High-Z	Outputs Disabled
H	X	X	High-Z	Deselected - Standby ($I_{SB)}$
V_{HC} (3)	X	X	High-Z	Deselected - Standby ($I_{SB1)}$

Figure 5: SRAM FSM

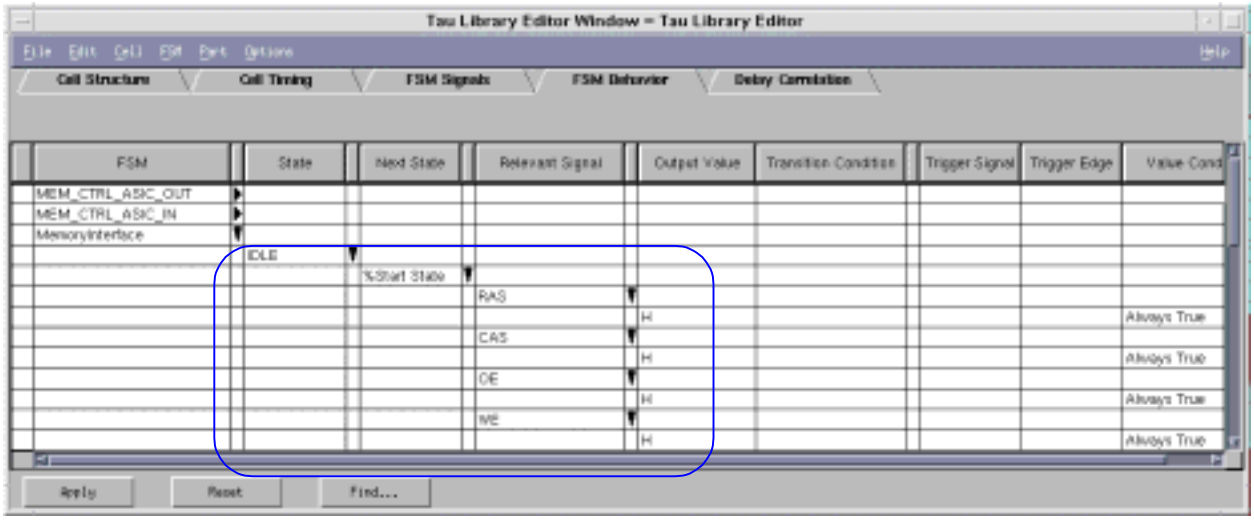


3. Next, if an FSM has an IDLE state determine if any of the output signals have known values (L or H) when in the IDLE state.

   Typically, control lines have a de-asserted value in the IDLE state while data lines have unknown values. If there are one or more output signals with known values in the IDLE state, then create a "%Start State" transition for the IDLE state. Under this transition, enter as relevant signals those *output* signals whose value is known in the IDLE state. Also, enter the known value for each of these relevant output signals.

   The timing diagram in Figure 2 indicates that the values on RAS, CAS, WE and OE are high when the memory controller starts a DRAM read access (note that the values on MA and MD are unknown). Accordingly, a %Start State transition is inserted for the IDLE state on the Memory Controller FSM. The relevant signals for this transition are RAS, CAS, WE and OE. The value defined for each of these signals is “H”. Figure 6 shows the relevant signals and the defined values in the FSM Behavior sheet of the library editor.

Figure 6: Relevant Signals of DRAM Read



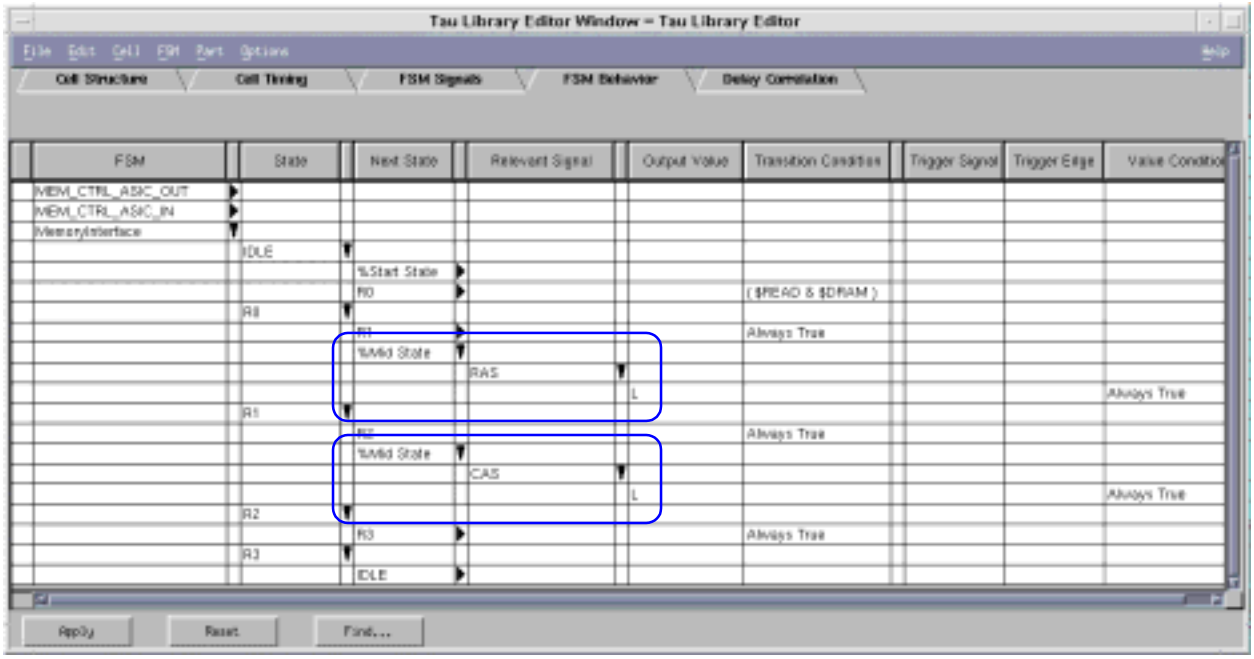
The truth table in Figure 3 shows that the value on IO (the only output on the SRAM) is unknown when the SRAM is in the idle state. Therefore, there is no need to enter a %Start State transition for the SRAM FSM.

4. Next, enter the state-transitions for each state on the FSM by entering the list of next states for a state.

   A %Mid State transition describes behavior exhibited by an FSM when in the middle of a state. For an asynchronous FSM this transition is qualified by the trigger signals, and optionally trigger edges, which specify the transitions on its inputs that an FSM responds to when in the middle of a given state. Trigger signals and edges are not required for %Mid State transitions on synchronous FSMs because the trigger signal is known to be the clock signal.

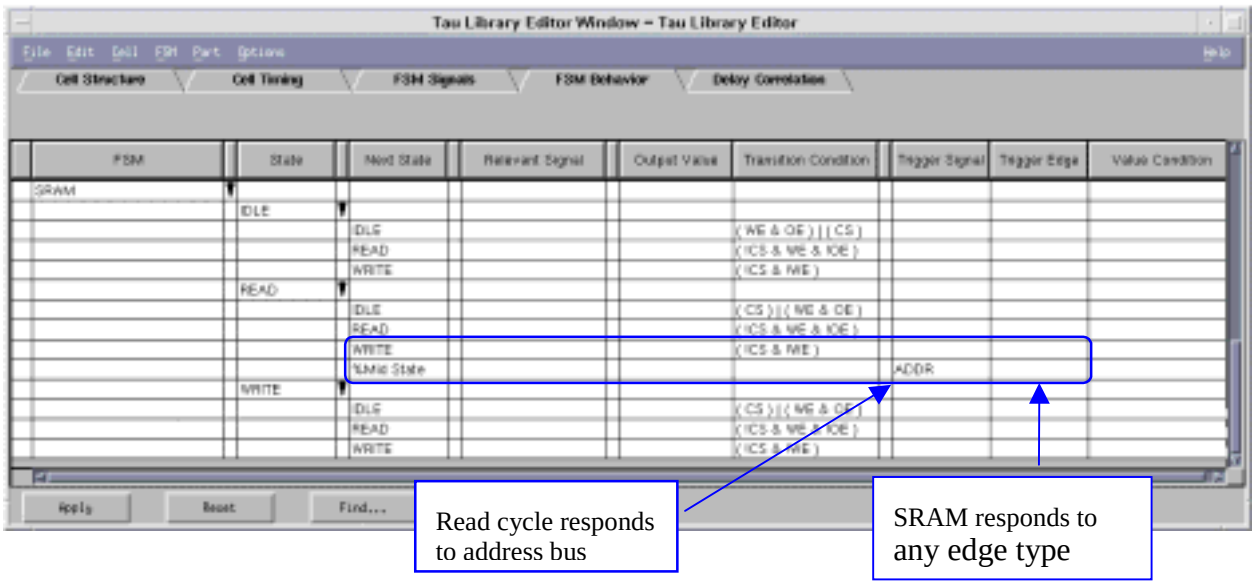
   For the timing diagram in Figure 2, each state transitions to exactly one next state. The IDLE state transitions to state R0, R0 transitions to R1, R1 transitions to R2, R2 transitions to R3, and R3 transitions to IDLE. Both the R0 and R1 states have %Mid State transitions because RAS and CAS are asserted in the middle of these states (at the falling edge of clock).

Figure 7: Next States and Mid State of DRAM



For the SRAM FSM, each state could potentially transition to any other next state as defined by the truth table in Figure 2. Therefore, the Next States for the IDLE, READ and WRITE states are IDLE, READ and WRITE. In addition, during the read cycle the SRAM responds to transitions on the address bus, so you should enter a %Mid State transition for the READ state, whose trigger signal is ADDR. The SRAM responds to any type of transition on the address bus, and so it not necessary to specify a trigger edge (rise, or fall) for the ADDR Trigger Signal. Figure 8 shows the Next States and %Mid State information of the SRAM FSM.

Figure 8: Next States and Mid State of SRAM



5. For each state-transition that is not of type %`Mid State` or %`Start State` enter its state-transition condition.

   If a state can transition to multiple next states then you need to specify state-transition conditions for each of these state transitions. These conditions must be non-overlapping (a state cannot enter two or more states under the same condition) and complete (it should not be the case that under some condition the next state entered is unknown).

   For the timing diagram in Figure 2, each state transitions to exactly one next state. For this reason, the state-transition conditions are always true. Figure 7 shows this for all read transitions, R0 to R1 etc.

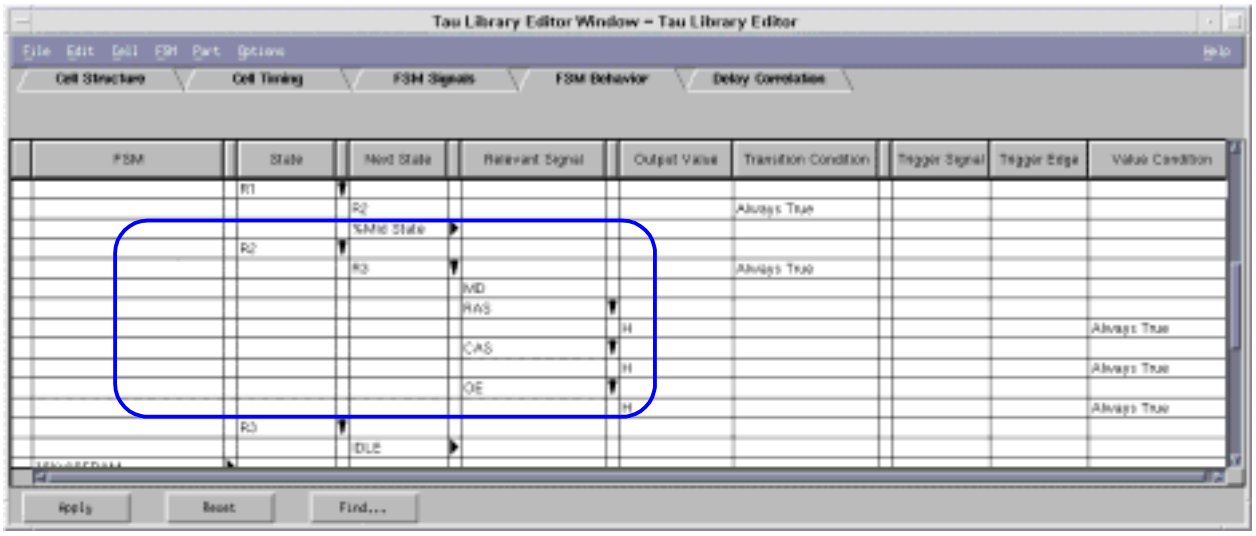
   The SRAM FSM state-transition conditions are defined by the truth table in Figure 4. Figure 8 shows the conditions entered in the FSM Behavior sheet.

6. For each state-transition, define the list of relevant signals.

   Relevant signals are input signals that are latched and output signals whose values are changed when a state-transition occurs.

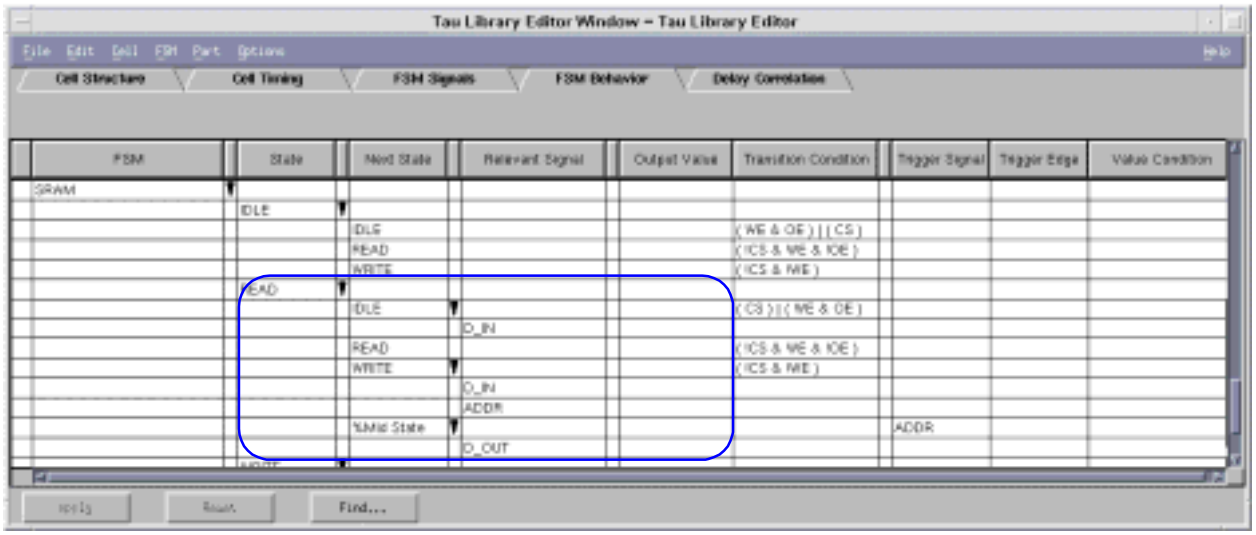
   The timing diagram in Figure 2 reveals that the RAS, CAS, OE and MD signals change relative to BCLK on the R2 to R3 transition. RAS, CAS and OE change to high and MD must be valid for 4 additional ns. Figure 9 shows how you enter these signals in the FSM Behavior sheet. Relevant Signals are inserted for the signals and the Output Values are set accordingly.

Figure 9: Relevant Signals of DRAM Read Cycle



The relevant signals for the SRAM Read State are shown in Figure 10.

Figure 10: Relevant Signals of SRAM



7. For each relevant output signal, define the value placed on the output signal once the state-transition occurs. When a relevant output signal can take multiple values, then enter the condition when each of these values are placed on the output. Like state-transition conditions, output-value conditions must be non-overlapping, though they need not be complete. This reflects the fact that it is permissible for the value on an output signal to be unknown, or placed in High-Z, under some condition.

   For processors etc., you sometimes cannot express the value driven onto data and address busses as a function of values on inputs to the processor. In this situation, you need to enter an internal signal to the FSM that is not connected to any port on a cell, but which reflects information obtained from the internal circuitry of a cell. You enter an internal signal in the same manner as any other signal – Tau recognizes it as internal by virtue of the fact that it is not connected to any port.

   Figure 11 shows a simple example of an output enabled buffer FSM. The output value, “in” is placed on the relevant signal “out” when the condition is “!oe” (output enable low).

Figure 11: OE Buffer FSM