EMA TimingDesigner

The industry's most accurate static timing analysis

Digital Timing

Using TimingDesigner to Develop State Machine Architecture for Complex Interfaces

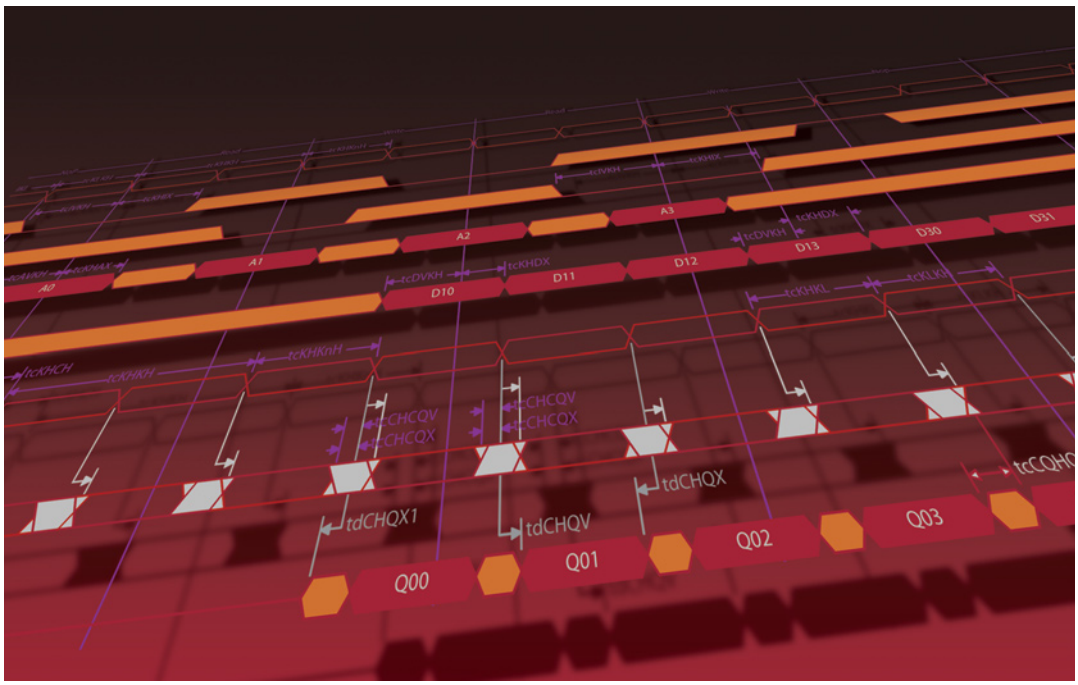
Jerry Long – Product Marketing Manager, EMA Design Automation
Sandy Helton – Principle Engineer, Xilinx Inc.

Learn about:

- Using TimingDesigner® Derived Signal features for development of one-hot state machines
- Using Waveform "Value Decode" features for state naming directly from bus values
- Validating input equation alterations quickly and easily using your HDL language of choice

Revision:

- Revised: 03/12/2008
- For use with: TimingDesigner 9.0 and higher



EMA Design Automation™
ema-eda.com

Table of Contents

Table of Figures

Introduction

"The unique features contained in TimingDesigner provide an environment for not only deciphering interface timing, but also allow insight into data manipulation issues beyond the interface."

TimingDesigner from EMA Design Automation has many powerful features that allow a multitude of timing and verification issues to be analyzed. Many companies employ TimingDesigner to aid in complex interface design and development as it provides an easy, self-intuitive method to address static timing issues using timing diagrams. One such area is complex memory interface design and control. This paper presents some TimingDesigner 'use model' ideas in common use today for development of Double-Data Rate (DDR) interface and controller design.

Designing controllers for DDR and/or Quad Data Rate™ (QDR™) memory devices presents some very unique timing challenges such as signal skew analysis and adjustments, and specific clock generation and control, all to ensure proper clock/data relationships. Typically DDR style controllers are implemented in FPGA devices due to their register-rich architectures, powerful PLL clock network generators, and memory interface friendly I/Os, all providing a highly efficient and flexible (i.e. programmable) environment to read, write, and process data. These memory interfaces are typically operating at frequencies of 200 MHz and higher, presenting non-trivial timing challenges.

However, the interface timing is not the only complex issue at hand, there's also the challenge of data management and manipulation within the FPGA fabric. Depending on the nature of the controller design at hand, the data preparation phase can be several magnitudes more complex than data capture. The complexity involves how to feed data processing units, most of which are completely independent from one another and can therefore be operating at different clock rates.

The unique features contained in TimingDesigner provide an environment for not only deciphering interface timing, but also allow insight into data manipulation issues beyond the interface. Using TimingDesigner in conjunction with an industry standard HDL based simulator provides a methodology for functional verification and debugging of very complex design issues much faster than simulation environments alone. TimingDesigner can play a very prominent role in the identification and resolution of system operation and control issues, providing very high observability at a low-level of abstraction.

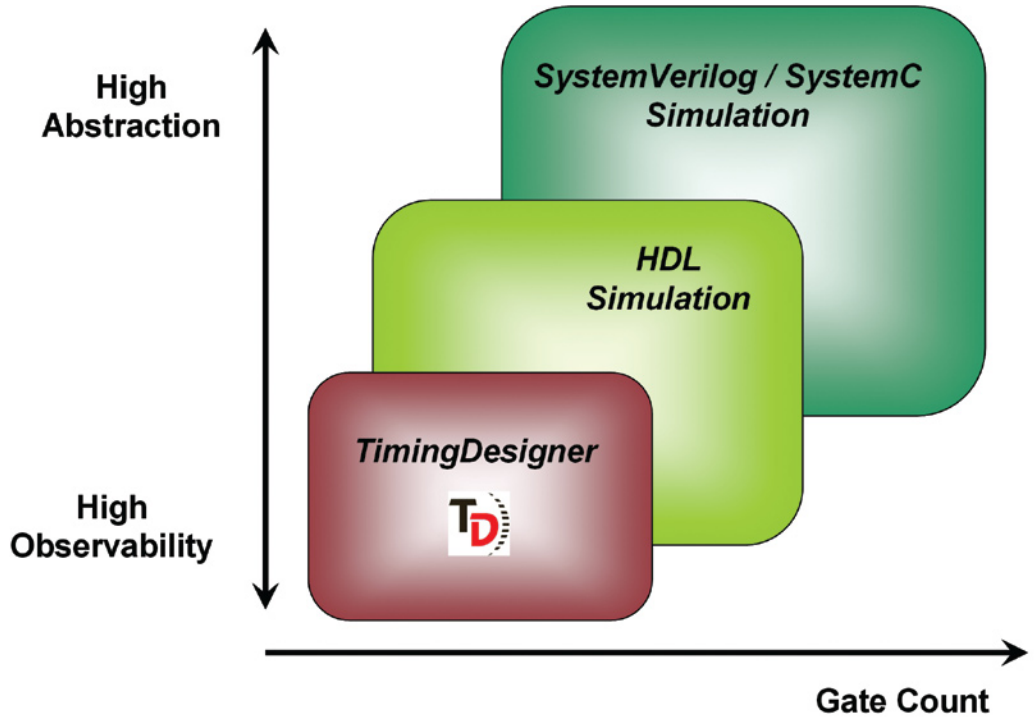


Figure 1: TimingDesigner provides high observability at a low level of abstraction

Memory Controller Design Challenges: State Machines

"TimingDesigner's Derived Signal and Derived Clock features provided a monumental step forward for one-hot state machine design tasks and the problems they present…"

The advent of DDR style memory devices was necessary to overcome the bottle-neck in data processing paths: the access to the memory storage device itself. Data capture on both rising and falling clock edges of synchronous memory devices effectively doubles the data access rate and allows data processing engines the luxury of a more continuous data stream. As is usually the case when a substantial design challenge has been resolved, DDR memory opened the door to even more powerful controller applications with a voracious appetite for data. One very effective method for controlling these data processors is the use of synchronous state machines.

State machines are very powerful control mechanisms that can be designed with various configurations and styles, ranging from the very simple (a counter for example), to the very complex (a DDR memory controller and data processor). Due to the nature of complex state machines and their dependency on input signals as well as current state, they are easiest to implement with HDL conditional constructs such as IF-THEN-ELSE, and CASE statements.

A topic of concern with state machine implementation is how the machine is encoded. The most common method of encoding a state machine is binary encoding where the total number of states can be encoded into a minimally sized bus structure (2N represents the total number of states and 'N' represents the encoded bus size). However, there is a trade off with binary-encoding in that the minimal amount of registers used typically require rather large decode logic equations for each next state event. Large decode logic translates to "wide" P-term equations which can consume quite a bit of timing latency.

One-hot encoded state machines use a single register for each state representation, and while this means there will be a higher number of registers in use, the encoding required for each is minimal. Minimal register encoding translates to "narrow" P-term equations for each state bit. Depending on the complexity of the machine and the number of states, the amount of registers required could quickly become quite large. But this is typically not a problem with the FPGAs of today as their register counts are quite large. In addition, FPGAs have limited inputs to the configurable logic entity: usually a 4 to 6 input Look-Up Table (LUT). So one-hot encoding, with its smaller logic levels and higher register counts, is the most efficient and therefore best suited style of state machine implementation for FPGAs.

One-hot state machines do have some pitfalls to be aware of. Since each state is represented with just one bit active at any time, the collective bits that represent the state machine vector do not fully enumerate the states available, so care must be taken for recovery of illegal states (more than one state bit active at a time). The reset condition must be considered as well as any next state decode latency that exceeds the clock period. All of these issues are conditions that can cause the state machine to enter an undefined state and therefore must be tested for in the design process.

TimingDesigner as a State Machine Design & Analysis Aid

Powerful HDL Feature Set

TimingDesigner is an extremely versatile tool that provides designers with enough flexibility to implement static timing analysis for any digital interface protocol, experiment with various device configurations affecting timing performance, and provide incremental functional verification of design segments. With powerful features such as Verilog and VHDL compliant Derived Signals, Derived Clocks, and other patented processes, TimingDesigner provides users with the ability to prove out design ideas, verify existing design elements, and debug problem design areas very quickly.

TimingDesigner's Derived Signal and Derived Clock features provided a monumental step forward for one-hot state machine design tasks and the problems they present, especially since existing HDL design code can be used directly to emulate and model various state machine transitions and associated data path control operations. HDL support is key here since it's a common language for FPGA and ASIC designers and therefore provides easy translation into TimingDesigner analysis tasks.

Validating State Machine Design

To describe how a state machine design can be monitored and validated, we'll focus on the Verilog HDL language and concatenate the state bits into a TimingDesigner Derived Signal, creating a vector that provides continual status of current machine conditions. For a large group of one-hot state bits this can be an incredibly complicated code statement to assemble, but TimingDesigner's ability to nest Derived Signals together allows designers to build the state bits up individually (or into several small groups) and concatenate them all together to present the information as a single vector. This is of tremendous value to designers.

In addition, the Value Decode option available for all TimingDesigner waveforms allows a unique mnemonic label to be assigned to each state for easy state identification within the diagram. These human readable vector names simplify state machine analysis and debug by reducing the effort level required to identify bit presentation for each valid machine state. As the complexity of the state machine increases, typically so does the number of states involved and the more valuable this feature becomes.

TimingDesigner also provides the ability to hide waveforms and other diagram events. Each individual state vector can be hidden in lieu of the concatenated result which makes for a very clean analysis environment, and allows a sense of hierarchy so that designers are not overly concerned about the minute details that make up the state itself. This allows them to concentrate on the more important aspects of the machine operation such as state-to-state transitions.

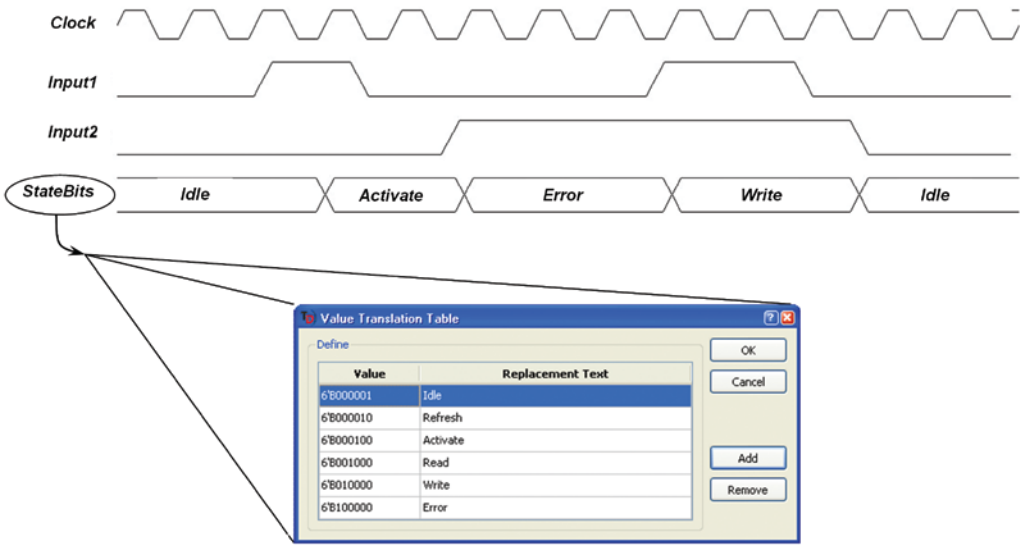


Figure 2: With the 'Value Decode' option, TimingDesigner can display mnemonic state names for easy reading

"TimingDesigner allows designers to very rapidly find issues that are causing problems, implement any input changes to correct them, and then test… all in a matter of minutes."

Providing Diagnostic Aid

Since the Derived Signals are HDL compliant, the actual synthesized Verilog code for the state machine can be used to observe exactly what the design is implementing and provides the ability to debug specific issues. For example, during simulation regressions one may encounter erroneous issues with state transitions whose causes aren't entirely obvious, but their characteristics point to some areas of suspicion. With TimingDesigner, these exact scenarios can be implemented quickly to achieve consistent results with simulations allowing one to pinpoint exactly why this undesired phenomenon occurred. Once the cause for the behavior is indentified, correctional state machine changes for the situation could be implemented directly in TimingDesigner to see immediate results and eliminate the problem.

As a diagnostic aid, TimingDesigner helps to replicate what is viewed in simulation, and then allows quick input signal changes to verify that the issue is indeed resolved, as well as exposing any residual affects of the change. Solving problems such as these with a simulator alone could take up to an hour or more due to the required setup necessary for input stimulus and circuit isolation, just to get to the point of exposing the cause of the issue. TimingDesigner allows designers to very rapidly find issues that are causing problems, implement any input changes to correct them, and then test to see if those changes not only correct the issue, but also didn't adversely affect the design, all in a matter of minutes. Simulation should still be done for regressing design integrity, making sure it's all correct, but TimingDesigner essentially provides a more rapid solution to these complex control issues.

Validating Data Path Management

Some designs require very complex management methods for data paths. Large bus structures may need to be split apart, rearranged, and then re-assembled. For instance, a Distributed Rule Engine (DRE) might take a 128-bit data domain and re-organize it into multiple 64-bit domains. Information is typically read out of a data pipe and then parsed into separate fields so the state machine can respond appropriately. TimingDesigner provides the ability to directly decode a vector field, pulling 2 or 3 bits out of a very long128-bit vector, and use that information in the state machine just as is done in the actual Verilog design code. TimingDesigner has numerous HDL code features that allow design analysis to parallel the development of design code, saving valuable design and debug time.

Perhaps one of the most difficult design aspects with data path management is state machine handshaking between two data channels that have asynchronous FIFO interfaces; a situation where one data stream is fed from two (or more) independent data streams that have different clock rates. The design must correctly manage data alignment and propagation as well as the complexities of changing bus sizes, all while maintaining consistent data flow. Making sure data is delivered correctly and accurately is a very complex design problem and is another example where TimingDesigner is very handy.

With these types of asynchronous interfaces, data is read from a FIFO then written to a data channel that can throttle the data rate up or down at any given clock cycle. The data channel is essentially data pipe-lines with a "sender" pipe and a "receiver" pipe, both mediated by an intermediate pipe-line. Each sender and receiver pipe can, without advanced notice, become empty and simply stop, or one could be full and therefore not take any more data. So they must be carefully managed such that any impedance of data flow is communicated back to the appropriate pipe to suspend their actions. The control process will take time (typically a full clock cycle or more) to occur and therefore requires accommodation of any extra data coming in. This is a very tricky thing to do well and causes both sender and receiver circuitry to turn off and on frequently.

"In a matter of minutes they can observe how it affects the system without having to set up a simulation for each write point possibility."

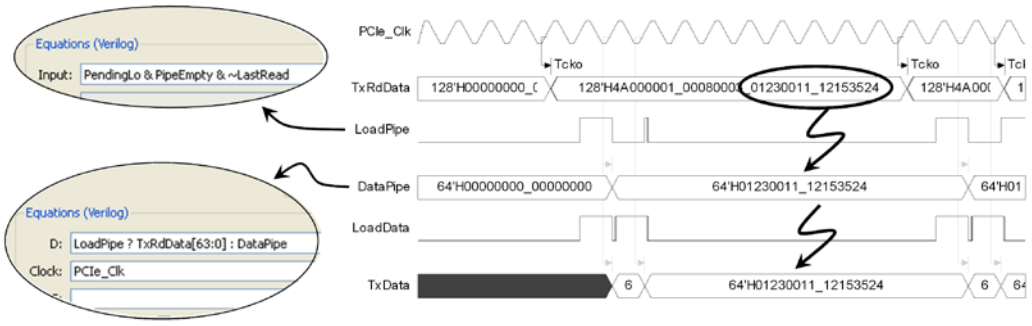


Figure 3: TimingDesigner allows HDL equations that can directly manipulate elements of a data bus and can provide quick verification of problem resolutions

These designs require such careful planning for the point at which the FIFOs are written that it becomes tremendously helpful to experiment with the timing controls in the design for optimum operation. TimingDesigner can provide instant visibility for the effects of various timing options allowing designers to alter the point at which data is written and observe how the data pipes and their respective control systems react. Using Derived Signals, designers can alter signal inputs to effect sample data points (when data is to be written to a FIFO) and manually pan those through a variety of write point possibilities and immediately observe how each one affects the system. Of course, provisions must be made to ensure that all input conditions are consistent with system operation outputs. This capability quickly allows designers to pan edge events affecting the operation of state machines, modeling many different input scenarios—if the signal occurs early, late, or coincident with another event. In a matter of minutes they can observe how it affects the system without having to set up a simulation for each write point possibility, and numerous corner cases can be evaluated as well without having to invent each one in a simulation.

With the powerful features of TimingDesigner's Derived Signals, state machines can be designed as extremely complex or moderately simple, and give extraordinary capability for design accuracy as well as intimate control of all state machine inputs. And since Derived Signals are based on HDL language syntax, they can reference parts of other Derived Signals to provide a model for the exact transfer of data from one pipe to another and allow observation of the entire data propagation path.

Summary

Design engineers use various methods to develop their ideas: bubble-diagrams, pseudo code, spreadsheets, and charts. Utilizing interactive timing diagrams with TimingDesigner is another method that engineers can employ. While most aren't big on GUI type tool interfaces, the examples described here are cases where a GUI based visual approach is often the most effective. These situations are such that designers really don't want the complexities of creating unique stimulus files and have to apply and analyze the results in a simulation. Rather, designers would like to approach these situations with a very focused and narrow scope to resolve problems and observe specific situational behavior, and TimingDesigner is extremely well suited for just that.

TimingDesigner's numerous features such as HDL compliant Derived Signals and Value Decoding options for mnemonic state labeling are extremely useful for debugging as well as design experimentation. Modeling situations using TimingDesigner allows vision and control on just exactly what is needed to identify and resolve issues, as well as providing the luxury of experimentation to determine residual effects without having to return to the simulation environment, saving an enormous amount of debugging time.

Timing diagrams are also the best way to visualize what's happening within a system interface. It's always good to work through a new interface protocol by creating and referencing the correct waveform relationships between modules. Not only does it re-enforce one's understanding of correct operation, it allows team members to work directly from the specifications, and provides accurate documentation that can be used in functional specifications for proven interface protocol performance. TimingDesigner is the industry standard timing diagram tool designed just for these purposes.

For more information please visit: www.TimingDesigner.com.

Where to Go for Help

Support

EMA Design Automation is committed to providing unsurpassed customer service and support. Our support site includes an extensive portfolio of TimingDesigner information to keep you productive.

You'll get access to product downloads, a searchable knowledge base, online tutorials, training information, FAQs, user manuals and more. Please visit our website http://support.ema-eda.com/ to access more detailed information.

Support Contact Information

Technical Support.....techsupport@ema-eda.com
License Information.....update@ema-eda.com
Phone (within North America).....800.813.7494
Phone (outside North America).....585.334.6001
Fax.....585.334.6693

General Contact Information

EMA Website.....www.ema-eda.com
TimingDesigner Website.....www.timingdesigner.com
Information and Sales.....info@ema-eda.com
Jobs.....resumes@ema-eda.com

cadence™
CHANNEL PARTNER

EMA Design Automation™
ema-eda.com