

BDT: A Technique For System-level Modeling

David Jakopac, Sanjay Mehta, Satish Nagarajan, and Sowmitri Swamy

Vista Technologies, Inc.

1100 Woodfield Road

Schaumburg, IL 60173

Phone (708) 706-9300

FAX (708) 706-9317

Email: David-Jakopac@VistaTech.com

ABSTRACT

Behavioral Design Technique (BDT) is a new methodology for system-level design and modeling in VHDL. BDT is a graphical, event/action-flow modeling technique that encompasses a wide variety of behaviors including, but not limited to, system-level modeling.

Typical behaviors of entities in the digital domain have several separable and distinct threads of control. These threads of control interact with each other in a well-defined manner. Threads describe concurrent activities. In BDT, the thread is the primary mechanism for describing the flow of control.

BDT is a narrative form of modeling. It formalizes the way an engineer might technically describe a piece of hardware to another engineer. BDT models can be understood and described thread-by-thread.

A thread is a sequence of events. It sequences through these events and then repeats. A set of actions can be associated with each event on the thread. Actions in one thread can cause events that another thread is dependent upon.

We call the sequence of events on a thread the event-flow and the actions associated with a specific event its action-flow. This captures the essence of most models of hardware. There is an overall organization level based upon events and what-happens-when. The detailed operation falls below this level into the action-flow. This makes BDT ideal for system-level specification. The overall organization of a behavior can be specified with the event-flow.

We will describe a BDT editor for the creation of BDT diagrams and the generation of VHDL code. Several BDT examples, including a system-level model of a RISC processor and a more detailed model of its pipelined control unit, will be presented.

BDT: A Technique for System-Level Modeling

October 19, 1992

David Jakopac

Vista Technologies, Inc.

1100 Woodfield Road, Suite 108

Schaumburg IL 60173-5121

Phone (708) 706-9300

FAX (708) 706-9317

David-Jakopac@VistaTech.com

and

Sanjay Mehta, Satish Nagarajan, and Sowmitri Swamy

Vista Technologies, Inc.

Vista Technologies, Inc.

David Jakopac

BDT

Why BDT?

Motivation

- Hardware is a collection of concurrent, interdependent operations.
- To describe behavior we must specify when operations occur and what operations happen
- To specify when an operation occurs, we need a reference: time, or more commonly, the occurrence of a specific value in some signal. We call such references events
- Specifying what operations occur is known as an action.
- BDT is an event-action model for specifying behavior.

Vista Technologies, Inc.

David Jakopac

What is BDT?

- Modeling is the intricate process of formalizing human thought and understanding of hardware, and BDT is a narrative form of modeling.
 - The model formalizes the way one person might technically describe a piece of hardware to another.
- BDT organizes events and actions into linear sequences called threads. Very often a thread, by itself, constitutes a sub-behavior, aiding comprehension of the model.
 - Threads are easy to construct due to extensive graphical support.
- The collection of threads constitutes the behavior of the system.
 - Models can be described and understood thread by thread.

David Jakopac

Vista Technologies, Inc.

Technology Background

- Event/response descriptions have been proposed as a formalism to design Graphical User Interfaces.
- University research has proposed such models as front ends to (high level) behavioral synthesis.
- Other researchers have used similar techniques in a limited context (e.g., interface logic design).
- BDT breaks new ground by generalizing the basic event-response paradigm to include internally generated events.
- The notion of an event, an action, and the semantics of their interplay have been refined and formalized.

David Jakopac

Vista Technologies, Inc.

How BDT Works

Threads

- Smallest unit of concurrency
- Waits for each event on the thread in order
- Each event may, or may not, have associated actions
- Events are not lost by processing actions
- Easy to associate a thread with a hardware operation
 - Example: a counter with an asynchronous ripple carry output has two threads, one for counting and one for the ripple carry

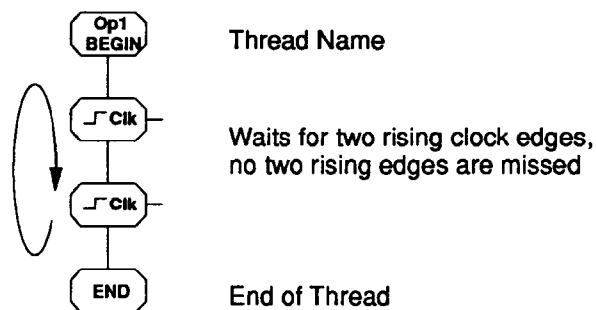
David Jakopac

Vista Technologies, Inc.

How BDT Works

Event flow

- Events on the thread form the event flow, the overall organization of the task the thread performs
- Events are sequential and are caused by changes in values to signals or events generated in other threads
- Can specify timeouts and delays in the event flow



David Jakopac

Vista Technologies, Inc.

How BDT Works

Actions

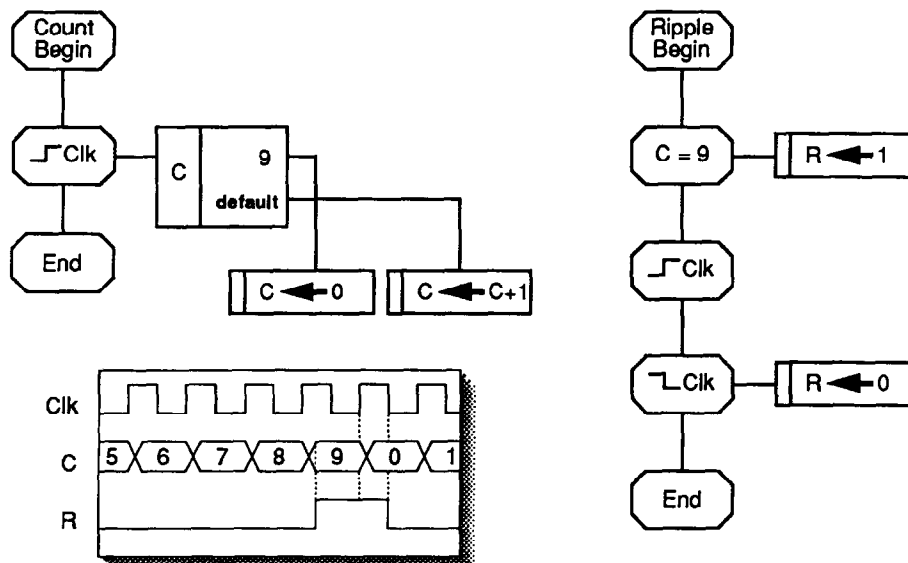
- Actions are associated with an event in a thread
- Actions can effect data (e.g., assignment), generate new events, and effect the event-flow in other threads
- Data objects belong to a universal class of objects (no signal/net or variable distinction)

David Jakopac

Vista Technologies, Inc.

How BDT Works

A simple example—BCD counter with ripple carry output



David Jakopac

Vista Technologies, Inc.

How BDT Works

Action Flow—Generation of HDL code

- BDT supplies high-level modeling concepts for which there is no direct support in HDLs
- Thread operations (abort, suspend/resume) generate sophisticated and, potentially, complex code
- Automatic determination of signal/net versus variable
- Thread could translate to wait-filled process statement or state-machine oriented process
- Code from multiple threads can be merged under certain circumstances

David Jakopac

Vista Technologies, Inc.

Summary

BDT provides a powerful natural approach to hardware modeling

- models hardware as a collection of concurrent, interacting behaviors.
- synchronous and asynchronous behaviors are treated uniformly.
- plenty of expressive power:
 - for behaviors such as indefinite suspension, mutual exclusion, self-abort, suspend-resume, daisy chaining, hand-shaking, delay timing, timed waits, and time-outs.
- groups of threads can represent specific components supports re-use of models

David Jakopac

Vista Technologies, Inc.