Specification and Synthesis of Bounded Indirection

M. Esen Tuna  Kamlesh Rath†  Steven D. Johnson
Indiana University Computer Science Department
Bloomington, Indiana 47405, U.S.A.
{mtuna,rathk,sjohnson}@cs.indiana.edu

Abstract

Bounded indirection is a restricted form of pointers, for system specification. It provides a mechanism for compact descriptions of many complex control structures, such as interrupts, continuations, and dynamic connections between machines. We describe three kinds of indirection — control state, value and net indirection — for use in different aspects of system description. Transformations on indirection representations and methods for synthesizing bounded indirection within the framework of behavior tables are presented.

1 Introduction

Hardware description languages have evolved from programming languages with many of their features and are extended with additional constructs for hardware specification. Some HDLs are primarily simulation languages with synthesizable subsets, whereas others are primarily used for describing synthesizable hardware. In this paper, we introduce the notion of bounded indirection, a restricted form of pointers, as a synthesizable hardware description construct. It can model complex control structures as simple data-path descriptions, thus enabling a designer to write thus more compact system descriptions. It can also be used to model selection mechanisms for connections between sequential components in a system.

Many hardware description languages have two dialects, one for structural descriptions and information, another for behavioral descriptions and control flow. Most synthesis tools based on such languages use separate data structures — control-flow graphs and data-flow graphs as their internal representations to perform optimizations. These different internal representations restrict designer’s ability to explore the design space by obscuring the links between the behavioral and structural aspects.

Indirection bridges the distinction between behavior and structure. However, to express indirection effectively, we need a notational framework to represent both behavioral and structural facets of a design. Behavior tables [1] provide us with such a framework.

Behavior tables are an extension of register transfer tables that can model control, datapath, protocol, and data abstraction facets of a system. A behavior table is a representation of a finite state machine that models system behavior. Each row in a behavior table represents a transition in the machine described by the table. The columns are divided into two sections, the decision section and the action section. The decision section represents the state and conditions that must hold for the transition to be executed. The action section represents the data flow through the functional units, ports, and the next state.

1.1 Related Work

VHDL [2] is emerging as a standard among hardware description languages, with a large number of language constructs, not all of which are synthesizable. Access type and selected name constructs in VHDL are similar to the bounded indirection construct discussed in this paper. But, these and other pointer constructs in VHDL cannot be synthesized by most tools [3, 4]. Some aspects of behavior tables are similar to Cheng and Krishnakumar’s extended finite state machine model [5], and Drusinsky and Harel’s statecharts [6], but these finite state machine models do not support indirection.

Bounded indirection is a synthesizable structural construct, that can be added to any HDL to enhance the expressiveness of the synthesizable language. We describe a method for synthesizing bounded indirection and define transformations on this construct. We also present three examples to illustrate the different uses of bounded indirection. The research reported here grew out of our design derivation methodology, which is based on first order functional algebra [7].

---

*Research reported herein was supported, in part, by NSF: The National Science Foundation, under grants numbered MIP 89-21842 and MIP 92-08745.
†Kamlesh Rath is currently a Senior Member of Research Staff in the VLSI and Video Communications Group at Philips Research Laboratories, Briarcliff Manor, NY 10510.

1066-1395/95 $4.00 © 1995 IEEE
2 Control State Indirection

Behavior tables allow for state identifiers to be stored in registers. These can then be loaded as the next state, and used as a branching address. References to a register name in the next state column in a behavior table, instead of a specific state identifier, is called control state indirection. The referenced register must hold a state identifier. State indirection is a useful mechanism to model interrupts, continuations, and procedure calls.

Table 1: Interrupt modeling

Table 2: Behavior table for the interruptable DMA controller (Action Section)

Table 3: DMA controller (Decision section)

Figure 2: State indirection transformations

Figure 3: Interruptable DMA controller state diagram

Using behavior tables (Table 2,3) we can specify the controller in 4 states and 10 transitions, instead of 5 states and 15 transitions in a classical finite state machine model. Compact interrupt representation allows significant reduction in the complexity of the specification. The schematic of the synthesized state machine for the DMA controller is shown in Figure 4. A binary representation of the state identifiers and truth values in the decision section of the behavior table (Table 2,3) are used to create a boolean decision table, which can be synthesized into the command generator by logic synthesis. The rest of the behavior table can be synthesized using sequential synthesis techniques [8].
3 Value Indirection

A simple construct in behavior tables can be used to specify that the value assigned to some signal is selected from a bounded set of signals based on the indirection value in another signal. Table 4 shows the use of $\text{Ptr}$ as an indirection pointer. References to a register/port/signal $r$ is written as $\text{r}$, and the prefixes $\text{G}$, $\text{S}$ denote input and output dereference respectively. A reference to $\text{R1}$ or $\text{R2}$ is stored in $\text{Ptr}$ and used in the next transition to load $\text{R1}$ or $\text{R2}$ into $\text{R9}$. Figure 5 shows a fragment of the state diagram and the corresponding schematic for the behavior table shown in Table 4.

![Figure 5: Value Indirection](image)

Modeling this mechanism in the classical finite state machine model or in a language without indirection would require $n$ transitions or $n$-way case statement for $n$ possible references in the reference set $I_1$, instead of a single transition in the behavior table. Further levels of indirection can be added to the system if needed for a more compact representation.

A bounded value indirection can be transformed into a set of transitions, each with a decision for a possible reference value. The equivalent transition set must have a transition with every possible dereference value as a decision, and all other values of the corresponding transitions must be equivalent. Figure 6 shows the transformation on value indirection.

![Figure 6: Value Indirection Transformations](image)

3.1 Example: FM9001 microprocessor

The FM9001 [9] has an internal register file with 16 general purpose registers any one of which can be used as the pc. Modeling all the different operations that can be performed on the different pairs of registers would require $16 \times 3$ transitions in a classical finite state machine model, but can be modeled by 3 transitions using indirection.

![Figure 7: FM9001 Register File Datapath Schematic](image)

Figure 7 shows the schematic for the register file datapath which is synthesized from the regs column of the behavior table specification of the FM9001. The register to be updated is selected by $\text{Spc}$ or $\text{S(rn op ins)}$ output indirections. The input to inc is selected from the register file using the input indirection $\text{G(rn op ins)}$, and the input to dec is selected using the input indirections $\text{Gpc}$ or $\text{G(rm op ins)}$. Details about the derivation and synthesis of an implementation of the FM9001 can be found in [1, 10].

4 Net Indirection

Net indirection provides a mechanism for the connection of an input (output) port to one of several output (input) ports depending on the indirection value on another port. This enables us to specify dynamic connections between components in a system, such as buses and switches.

Consider the net connections where an input port $a$ is connected to one of the output ports in the reference list $I_1$ depending on the indirect value in $p_1$, and the data output port $b$ is connected to one of the input ports in the reference list $I_2$ depending on the value of $p_2$. The schematic corresponding to these net indirection primitives is shown in Figure 8.

![Figure 8: Net Indirection Primitives](image)

4.1 Example: Bi-directional Bus

We now consider a general purpose bi-directional bus that provides port-to-port as well as broadcast channels for communication between ports in the system. Describing such a structure in a functional netlist specification would require creating and connect-
5 Conclusions

In this paper, we presented bounded indirectness, a hardware specification construct. By restricting the range of each indirect signal to a bounded set of references, we were able to make it a synthesizable construct. Bounded indirectness is similar to pointers in programming languages, but with the important difference that bounded indirect signals are defined over small predefined types of signals names, whereas pointers are defined over a broad range of values in a type class. The synthesizability of this construct and a clear description of its basis in a finite state machine model make it particularly attractive for inclusion into hardware description languages. The three examples presented here provide evidence for the usefulness of bounded indirectness.

References


