Symbolic Multi-Level Verification of Refinement

Stefan Hendricx Luc Claesen

IMEC vzw/Katholieke Universiteit Leuven
Kapeldreef 75, B-3001 Heverlee, Belgium
e-mail: Stefan.Hendricx@imec.be

Abstract

VLSI-system design can, in general, be characterized in terms of the step-wise refinement of intermediate solutions. Despite the fact that such refinements usually do not preserve time-scales, current formal verification approaches mostly start from the assumption that both specification and implementation utilize the same scales of time. Realizing the importance of being able to cope with differences in timing granularity, this preliminary paper proposes a symbolic methodology to verify that a low-level finite state machine is a refinement of a high-level finite state machine. To illustrate our approach, the step-wise refinement — and verification — of a simple microprocessor is presented.

1. Introduction

Independent of the application domain — ranging from mechanics to microelectronics — system design can, in general, be characterized in terms of a step-wise refinement of intermediate solutions. In the design of VLSI-systems, in particular, well-defined (and well-automated) refinement steps — such as partitioning, control- and data-path synthesis, logic optimization, etc. — progressively *compile* high-level specifications into physical implementations.

The ability to guarantee the correctness of such refinement steps plays a crucial — and self-evident — role in producing qualitative designs. Not surpisingly, formal verification has become one of the focal points in electronic hardware design [1]. The majority of the formal approaches that have been examined so far, however, mostly restrict themselves to proving the correctness of the precise input-output behaviour of an implementation with respect to a given specification. More specifically, it is assumed that both specification and implementation employ the same scales of time.

Whereas the previous assumption is valid for applications such as equivalency checking and finite-state-machine comparison — both well-established areas of research — it is not always applicable. Most often than not, refinement does not preserve time-scales during the design-process. On the contrary, timing-granularity usually tends to increase. At the algorithmic level, for instance, instruction execution may be described in terms of a single (algorithmic) clock-cycle. In the final implementation of the microprocessor, the same instruction may be executed during several physical clock-cycli — the number of which is not necessarily a constant parameter of the instruction set.

Realizing the importance of dealing with these differences in time-scales, researchers are starting to look into this matter. In [3], Hoskote *et al.* report on verifying *containment* of finite state machines. Basically, the goal of their approach is to demonstrate that an implementing FSM can perform all the behaviours defined by the specifying FSM. Or, in other words, that the specification is *contained in* — instead of merely *equivalent to* — the implementation. In [2], Genoe *et al.* present the *SFG-Tracing* methodology, a symbolic technique to formally check the weak-observable input-output behaviour of low-level implementations with respect to algorithmic Signal-Flow-Graphs. Key elements to this approach are user-defined relationships between well-chosen signals in the specification and spatial/temporal signals in the low-level implementation.

Armed with the practical experience gained from the SFG-Tracing methodology, we recently started to investigate the formal verification of finite-state-machine refinement. In this preliminary research paper, we propose a symbolic methodology to verify that a low-level FSM is a *refinement* of a high-level FSM. Underlying this symbolic approach is again the concept of user-defined relationships, this time between low-level and high-level states.

2 Our Methodology

Underlying our methodology is the intuitive notion that when a finite state machine M_r is supposed to be a refinement of another finite state machine M, there should exist a

0-7695-0104-4/99 $10.00

close relationship between a (non-empty) subset of states of M_r and the state-set of M. To better understand the nature of such a relationship, the various ways in which FSMs can be refined, must be considered first.

2.1 Finite-State-Machine Refinement

Suppose that $s_1, s_2, \ldots, s_n$ are states of M and that s_{r1}, $s_{r2}, \ldots, s_{rn}$ are states of M_r. We can then distinguish between the following refinements:

- A single high-level transition $s_1 \xrightarrow{\tau^h} s_2$ can be refined into a sequence of low-level transitions $s_{r1} \xrightarrow{\tau_1^l} s_{r2} \xrightarrow{\tau_2^l} \ldots \xrightarrow{\tau_n^l} s_{rn}$.

  In this case, there exists a direct relationship between the high-level states s_1 (s_2) and the low-level states s_{r1} (s_{rn}). Typically, this type of refinement can be associated with implementing the execution of a single instruction inside a microprocessor.

- A high-level transition-sequence $s_1 \xrightarrow{\tau_1^h} s_2 \xrightarrow{\tau_2^h} s_3$ can be refined into a set of low-level transition sequences $s_{r1} \xrightarrow{\tau_1^l} s_{r2} \xrightarrow{\tau_n^l} s_{rn}$, $s_{r1} \xrightarrow{\tau_2^l} s_{r3} \xrightarrow{\tau_n^l} s_{rn}$, $\ldots$, $s_{r1} \xrightarrow{\tau_{n-1}^l} s_{r(n-1)} \xrightarrow{\tau_n^l} s_{rn}$. Here, there exists a direct relationship between s_{r1} and s_1, between s_{rn} and s_3, and between the low-level states $s_{r2}, s_{r3}, \ldots, s_{r(n-1)}$ and s_2. In addition, the transition relations should satisfy

$$\tau_2^h = \tau_n^l \tag{1}$$

$$\tau_1^h = \tau_1^l + \tau_1^l + \ldots + \tau_{n-1}^l. \tag{2}$$

  Typically, this type of refinement is encountered when defining the instruction-set of a processor — i.e. in the generic *Fetch-Decode-Execute*-cycle, the high-level *Execute*-state is refined into multiple low-level states.

- A single high-level state can also be refined into a low-level sub-FSM. This type of refinement reflects the well-known principle of hierarchical finite state machine construction. In this case, each low-level state can be related to the single high-level state.

Starting from the principle that a relationship does exists between certain low-level states and the high-level states, the concept of a mapping function is introduced.

2.2 The Mapping function $\Psi_{Map}(\,)$

Let S_{High} and S_{Low} denote the high-level and low-level state-spaces respectively. In addition, assume that $S_{Map} \subseteq S_{Low}$ is the (non-empty) subset of all low-level states that can be related to the high-level states.

The *mapping* function $\Psi_{Map}(\,)$ is a function that maps — or relates — states in S_{Map} to states in S_{High}. $\Psi_{Map}(\,)$ should satisfies the following properties:

- $\Psi_{Map}(\,)$ maps each state in S_{Map} to *one* state in S_{High}.
- $\Psi_{Map}(\,)$ may map multiple states in S_{Map} to the same high-level state
- $\Psi_{Map}(\,)$ preserves the high-level outputs.

A fundamental problem is to determine possible candidates for $\Psi_{Map}(\,)$. Obviously, between a given low-level and high-level finite state machine, many functions can be defined that satisfy the above properties. Here, we assume that the user supplies the necessary mapping information, in a fashion similar to the user-defined mappings employed in the SFG-Tracing methodology.

Starting from the user-supplied relationships between S_{Map} and S_{High}, the different low-level paths or *trajectories* between states in S_{Map} can be examined.

2.3 State-Set-Bounded Trajectories

Definition 1: A trajectory $\Delta_{s_0 \to s_n}$ *of a finite state machine M is a single path* $s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_{n-1}} s_n$ *through M, starting in state* s_0 *and ending in state* s_n.
It should be obvious that such trajectories can be of infinite length. As such, the concept of trajectories is too general to be of real practical use. Therefore, we restrict ourselves to trajectories in which the source-state s_0 and the target-state s_n occur only at the beginning or ending of the path, but nowhere in between. We say that such a trajectory is *bounded* by the states s_0 and s_n.

Definition 2: A trajectory $\Delta_{s_0 \to s_n}$ *of a finite state machine is* bounded *by states* s_0 *and* s_n, *if* s_0 *and* s_n *appear only at the beginning or ending of the trajectory.*
In other words, a trajectory $s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_{n-1}} s_n$ is *bounded* by s_0 and s_n, if $s_1, s_2, \ldots, s_{n-1} \neq s_0$ and $s_1, s_2, \ldots, s_{n-1} \neq s_n$. It is important to note that s_0 may be equal to s_n — e.g. a single transition from a state s_0 to s_0 is a trajectory bounded by s_0.

Definition 3: Assume S *is a subset of states of a finite state machine M. A* State-Set-Bounded *trajectory or* SSB-*trajectory* $\Delta^S_{s_0 \to s_n}$ *of M is a trajectory* $\Delta_{s_0 \to s_n}$ bounded *by states* $s_0, s_1 \in S$ and *containing no other states of* S

To prove that a low-level FSM is indeed a refinement of a high-level FSM, we need to demonstrate that each low-level *SSB*-trajectory associated with the user-defined mappings, implies the existence of a high-level transition.

2.4 Definition of Refinement

Given a FSM M_r with state-space S_{Low}, a FSM M with state-space S_{High}, and a mapping function Ψ_{Map}, which maps states in $S_{Map} \subseteq S_{Low}$ to states in S_{High}, such that for all states $s_{Low_1}, s_{Low_2} \in S_{Map}$ there exist states s_{High_1}, $s_{High_2} \in S_{High}$, for which

$$\Psi_{Map}(s_{Low_1}) = s_{High_1}$$
$$\Psi_{Map}(s_{Low_2}) = s_{High_2}$$

M_r is called a refinement of M, if for each state-set-bounded trajectory $\Delta^{S_{Map}}_{s_{Low_1} \rightarrow s_{Low_2}}$ between states s_{Low_1} and s_{Low_2}, a high-level transition T_{High} exists between s_{High_1} and s_{High_2}, such that

$$Eval(\Delta^{S_{Map}}_{s_{Low_1} \rightarrow s_{Low_2}}) \Rightarrow T_{High} \tag{3}$$

With *Eval*( ), state-set-bounded trajectories are (symbolically) evaluated with respect to the stability conditions implied by the high-level FSM — i.e. during a state-transition, input-signals are assumed to be stable.

2.5 Symbolic expressions for $Eval(\Delta^{S_{Map}}_{s_{Low_1} \rightarrow s_{Low_2}})$

Using symbolic manipulation techniques and fixed-point calculations, it is possible to automatically derive Boolean expressions for $Eval(\Delta^{S_{Map}}_{s_{Low_1} \rightarrow s_{Low_2}})$. In this subsection, a concise overview is given of how these symbolic expressions can be obtained.

For simplicity, the discussion below restricts itself to finite state machines with only 3 state-variables. As such, each state s is represented by $s = (x_0, x_1, x_2)$ (x_0, x_1 and x_2 are Boolean variables). A finite state machine M can be defined by its next-state function — i.e. the next state is expressed in terms of the current state:

$$(x_0^{next}, x_1^{next}, x_2^{next}) = (F_0(x_0, x_1, x_2), F_1(x_0, x_1, x_2), F_2(x_0, x_1, x_2))$$

In addition, M can be defined by a transition-relation $\delta^1_{(x_0,x_1,x_2)}(y_0, y_1, y_2)$, which states the Boolean conditions to reach (y_0, y_1, y_2) in a single transition from (x_0, x_1, x_2).

The Adapted Finite-State-Machine M^*

In order to evaluate the *SSB*-trajectories of a finite state machine M, an adapted version M^* must be constructed first. This adapted finite state machine M^* differs only in one aspect from M — each state-transition in M starting in a state $s \in S_{Map}$ is replaced in M^* by a transition to s itself. All other transitions (and states) of M are preserved in M^*.

The *n*-steps transition-relation $\delta^{*n}_{(x_0,x_1,x_2)}(y_0, y_1, y_2)$

Starting from the adapted finite-state-machine M^*, we generalize the concept of the transition-relation. The *n*-steps transition-relation $\delta^{*n}_{(x_0,x_1,x_2)}(y_0, y_1, y_2)$ denotes the Boolean condition to reach state (y_0, y_1, y_2) starting from state (x_0, x_1, x_2), in a sequence of n *or* less state-transitions (assuming stability of inputs).

The generalized transition-relation $\delta^{*n}_{(x_0,x_1,x_2)}(y_0, y_1, y_2)$ can be defined recursively. First, the 1-steps transition-relations are well-known:

$$\delta^1_{(x_0,x_1,x_2)}(y_0, y_1, y_2) = [y_0 = x_0^{next}] \wedge [y_1 = x_1^{next}] \wedge [y_2 = x_2^{next}]$$

$$\delta^{*1}_{(x_0,x_1,x_2)}(y_0, y_1, y_2) = [y_0 = x_0^{*next}] \wedge [y_1 = x_1^{*next}] \wedge [y_2 = x_2^{*next}]$$

Next, we can express the 2-steps transition-relation in terms of the 1-step transition-relations:

$$\delta^{*2}_{(x_0,x_1,x_2)}(y_0, y_1, y_2) = \delta^1_{(x_0,x_1,x_2)}(y_0, y_1, y_2) \vee \left[\sum_{\forall(q_0,q_1,q_2)} \delta^1_{(x_0,x_1,x_2)}(q_0, q_1, q_2) \wedge \delta^{*1}_{(q_0,q_1,q_2)}(y_0, y_1, y_2) \right]$$

For n > 2, the *n*-steps transition-relation is defined by:

$$\delta^{*n}_{(x_0,x_1,x_2)}(y_0, y_1, y_2) = \delta^{*n-1}_{(x_0,x_1,x_2)}(y_0, y_1, y_2) \vee \left[\sum_{\forall(q_0,q_1,q_2)} \delta^{*n-1}_{(x_0,x_1,x_2)}(q_0, q_1, q_2) \wedge \delta^{*1}_{(q_0,q_1,q_2)}(y_0, y_1, y_2) \right]$$

Fixed-point Calculation

For a given finite state machine M (and state-set S_{Map}), the above recursive definition knows a fixed-point solution — i.e. there exists a certain value n_f, such that

$$\delta^{*n_f}_{(x_0,x_1,x_2)}(y_0, y_1, y_2) \equiv \delta^{*m}_{(x_0,x_1,x_2)}(y_0, y_1, y_2)$$

for all $m \geq n_f$.

This fixed-point solution enables us to further elaborate *Eval*( ), the function responsible for the symbolic evaluation of *SSB*-trajectories. Using $\delta^{*n_f}_{(x_0,x_1,x_2)}(y_0, y_1, y_2)$, a single expression can be derived for *all SSB*-trajectories associated with S_{Map}. For that purpose, we simply have to impose that (x_0, x_1, x_2) and (y_0, y_1, y_2) both belong to S_{Map}. When $\chi_{S_{Map}}(\ )$ denotes the characteristic function of S_{Map}, we finally obtain:

$$Eval(\Delta^{S_{Map}}_{(x_0,x_1,x_2) \rightarrow (y_0,y_1,y_2)}) = \delta^{*n_f}_{(x_0,x_1,x_2)}(y_0, y_1, y_2) \wedge \chi_{S_{Map}}(x_0, x_1, x_2) \wedge \chi_{S_{Map}}(y_0, y_1, y_2) \tag{4}$$

2.6 Symbolic Verification

Using expression 4, the verification problem at hand can be re-formulated. To verify that M_r is a refinement of M — under the user-supplied mapping $\Psi_{Map}(\,)$ — we need to prove the following implication.

$$\delta^{M_r,*n_f}_{(x_0,x_1,x_2)}(y_0,y_1,y_2) \wedge \chi_{S_{Map}}(x_0,x_1,x_2) \wedge \chi_{S_{Map}}(y_0,y_1,y_2) \Rightarrow \delta^{M,1}_{(\psi_{Map}(x_0,x_1,x_2))}(\psi_{Map}(y_0,y_1,y_2)) \quad (5)$$

Since all the components of expression 5 are symbolic, any symbolic manipulation package can, in principle, be used to check the validity of this implication.

3 Application Example

Our approach has been successfully applied to formally verify the step-wise refinement of a simple microprocessor, one which closely resembles Joyce's Tamarack [4].

At the highest level, our microprocessor can be described in terms of a finite state machine with only 2 states — i.e. *Idle* and *Run*. In *Idle*, the processor is awaiting user-input, whereas in *Run*, the instructions in the program-memory of the processor are executed (see figure 1).

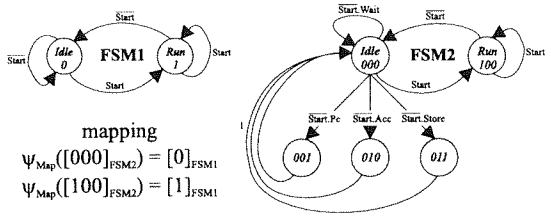

Figure 1. FSM2 is a refinement of FSM1

The *Idle-Idle*-transition of FSM1 can be further refined, by specifying the different input-modes of the processor — e.g. enter program-counter (*Pc*), enter data-word (*Acc*), store data (*Store*) or *Wait*. To verify that the resulting finite state machine FSM2 is a refinement of FSM1, we first define a mapping between low-level and high-level states (see $\psi_{Map}(\,)$ in figure 1). Based on this mapping-information, expression 5 can be constructed. Once constructed, we can positively verify that the refinement implication is satisfied.

In a similar fashion, we can establish that FSM3 — the result of replacing the *Run-Run*-transition of FSM2 by a generic *Fetch*[101]-*Decode*[110]-*Execute*[111] sequence — is actually a refinement of FSM2 (see figure 2).

In total, 10 different finite state machines — a sequence of gradually more detailed models — were derived for the controller of our microprocessor. Using our symbolic methodology, we were able to establish the correctness of each of these step-wise refinements.

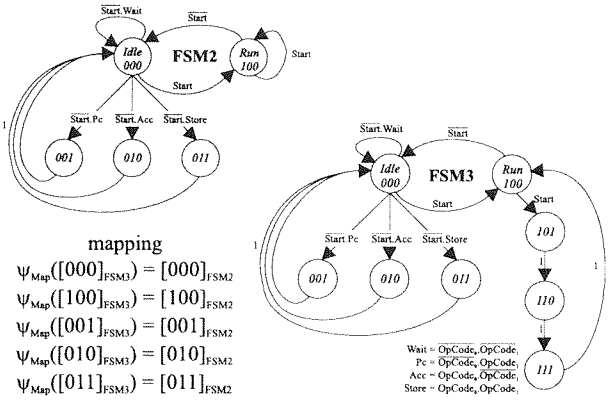

Figure 2. FSM3 is a refinement of FSM2

4 Conclusions

This paper introduced a methodology to symbolically verify the step-wise refinement of finite state machines. We have explained how symbolic manipulation techniques and fixed-point calculations can be used for that purpose. Key to our approach are user-defined relationships — the so-called *mapping*-functions — between the states of the state machines involved. To illustrate our approach, the step-wise refinement of a simple microprocessor was considered.

5 Acknowledgment

The research presented in this paper was supported by a scholarship from the Flemish Institute for the promotion of Scientific-Technological Research in Industry (IWT).

References

[1] P. Camurati and P. Prinetto. Formal Verification of Hardware Correctness: Introduction and Survey of Current Research. *IEEE Computer*, 21(7):8–19, July 1988.

[2] M. Genoe, L. Claesen, F. Proesmans, E. Verlind, and H. De Man. Illustration of the SFG-Tracing Multi-Level Behavioural Verification Methodology by the Correctness Proof of a High Level Synthesis Application in CATHEDRAL-II. In *Proc. International Conf. Computer Design (ICCD): VLSI in Computers and Processors*, Cambridge, MA, 1991. IEEE Computer Society Press.

[3] Y. V. Hoskote, J. A. Abraham, D. S. Russel, and J. Moondanos. Automatic Verification of Implementations of Large Circuits Against HDL Specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(3):217–227, Mar. 1997.

[4] J. J. Joyce. Formal Verification and Implementation of a Microprocessor. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis*, pages 129–157. Kluwer Academic Publishers, 1988.