Proc. CHARME (Conf. Correct Hardware Design and Verification Methods) 2005

Formal Verification of Synchronizers

Tsachy Kapschitz and Ran Ginosar

VLSI Systems Research Center, Electrical Engineering Department
Technion–Israel Institute of Technology, Haifa 32000, Israel
[ran@ee.Technion.ac.il]

Abstract. Large Systems on Chips (SoC) comprise multiple clock domains, and inter-domain data transfers require synchronization. Synchronizers may fail due to metastability, but when using proper synchronization circuits the probability of such failures can be made negligible. Failures due to unexpected order of events (caused by interfacing multiple unrelated clocks) are more common. Correct synchronization is independent of event order, and can be verified by model checking. Given a synchronizer, a correct protocol is guessed, verification rules are generated out of the protocol specification, and the model checker applies these rules to the given synchronizer. An alternative method verifies correct data transfer and seeks potential data missing or duplication. Both approaches require specific modeling of multiple clocks, allowing for non-determinism in their relative ordering. These methods have been applied successfully to several synchronizers.

1 Introduction

Large systems of chip are typically partitioned into multiple clock domains. Clock frequencies of the various domains and their relative phases may be unknown a-priori, and may also change dynamically [1]. Data transfers between different clock domains require synchronization [2]. When data enters a domain and happens to change exactly when the receiving register is sampling its input may cause that register to become metastable and fail [3]. This problem is mitigated by properly employing synchronizers. This paper describes methods for formal verification of synchronizers using model-checking [4].

Synchronizers are designed to allow certain time for metastability resolution. The amount of resolution time is determined according to the desired level of probability of failures. In this paper we assume that sufficient time has been allowed and no failures are expected. However, following metastability the synchronizer may resolve non-deterministically to either 0 or 1, and consequently proper synchronization is still not guaranteed. To mitigate that non-determinism, synchronizers are encapsulated in a bidirectional handshake protocol. The goal of formal verification is to guarantee correct execution of that protocol.

There are too many known synchronizer types and synchronization protocols, and it may be infeasible to define a single specification that could be used to verify all of them. Instead, we employ structural analysis to recognize synchronizers and to sort them into several a-priori known types. For each type, a set of properties has been defined, which, when proven to hold, guarantee correctness.

The paper describes how to generate formal verification executions of RuleBase (a model checker [5] using PSL [6]) for given synchronizers. We start with modeling of multiple clocks in Section 2. Next, in Section 3, we describe control verification method, based on converting the specification into PSL assertions. Data verification, which is not specification-dependent, is presented in Section 4. A more detailed description is given in [7].

2 Modeling multiple clocks

The model checker (MC) [5] performs its algorithms in a sequence of atomic *ticks*. Each synchronous component of the system being verified (the *design system*) is assumed to operate in atomic *clock cycles.* Common model checking assumes a single clock, but synchronizers must be verified while observing multiple clocks. Thus, we need to add special modeling of multiple clocks to our specification.

Clock modeling depends on how the clocks of the two domains are inter-related. If the two clocks are unrelated, they are modeled as two free variables. When the frequencies of the two clocks are assumed related by a rational number m/n (WLOG $m>n$) [8], then we specify to the MC that between any two edges of CLK2 there should be N active edges of CLK1, where $\lfloor m/n \rfloor \leq N \leq \lceil m/n \rceil$ (see [7]). A wide range of m/n ratios may be covered in a single execution of the MC if m/n is specified as a non-deterministic variable.

3 Control Verification

As stated above, data transferred between two mutually asynchronous clock domains are wrapped by a handshake protocol, implemented with control signals between the domains. We consider verification of the protocol by examining the control signals. The desired synchronizer handshake protocols are specified by means of STG (Signal Transition Graphs) that define the order of events (logic level transitions) in the synchronizer [9]. In this section we discuss how to convert the synchronizer STG directly into PSL assertions.

We first generate assertions to prove that if a signal transition event is enabled, it eventually happens. Each event has its own condition that enables its execution. In STG, the condition is fulfilled by a marking (a mapping of tokens to arcs) where all arcs incoming into the event carry tokens, enabling firing of the event. The condition is converted into a rule that verifies that the enabled transition actually takes place before the enabling state is changed [7]:

```
AG ( EnablingState(E) -> Transition(E) before !EnablingState(E) )
```

Next, we generate assertions that verify that events take place only when enabled:

```
AG (Transition(E) -> SetOfEnablingStates)
```

To verify that the given synchronizer complies with the specification STG, we prove the correctness of the constituent events with the above rules. The correct ordering of events is then implied by the ordering allowed by the STG.

4 Data Verification

Verifying the synchronizer control, presented in the previous section, is subject to two limitations: First, it is protocol specific--the rules depend on the specific STG and cannot in general be applied to other synchronizers. Second, the STG may need to be modified (e.g. to satisfy complete state coding [10]), in order to enable rule derivation [7]. In this section we present *data verification* of the actual data transfer, irrespective of the control handshake protocol. If the controller has an error, it will be discovered through data verification. The goal of data transfer verification is to prove that any data item sent by the sender is eventually sampled exactly once by the receiver.

The data transfer part of a synchronizer is shown in Fig. 1. The verifier interprets the loading of data DIN into the leftmost register as an attempt by the sender to send it. A sampling into the rightmost register is interpreted as an attempt by the receiver to receive data. The verifier must prove that no data item is either missed or sampled more than once by the receiver.

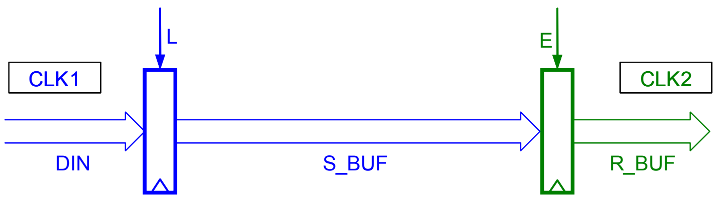

Fig. 1: Cross-domain data transfer structure

The first verification rule checks data integrity:

```
AG ( CLK1 & L & DIN(0)=1 ->
     next_event( CLK2 & E )( S_BUF(0)=1 ))
```

A similar rule can be written for the value `0`. Integrity is checked only for a single data bit because all the other bits will behave in the same way, as guaranteed by structural verification. In addition to data integrity, we should verify that:

- Data is not duplicated—the receiver does not sample the data if the sender did not send any:

```
AG ( CLK2 & E -> AX ( (CLK1 & L) before (CLK2 & E) ))
```

- Data is not missed—the receiver eventually receives data that was sent by the sender:

```
AG ( CLK1 & L -> AX ( (CLK2 & E) before! (CLK1 & L) ))
```

In words, between any two send events there must be one reception, and vice versa. The second assertion uses the strong *before!* operator (with !) to verify that the event (CLK2 & E) eventually takes place even if the subsequent event (CLK1 & L) does not happen at all.

5 Conclusions

We have demonstrated two methods for synchronizer verification using model checking. For *control verification*, a specification (in terms of STG) is employed to derive PSL assertions that are subsequently applied to the design. For *data verification* we seek correct data transfers (each sent data item is received exactly once) while ignoring the control operation. Both methods require specific modeling of multiple clocks, allowing for non-determinism in their relative ordering.

These methods have been applied successfully to a number of synchronizers, such as the two-flip-flop synchronizer, a dual clock FIFO, and an Adaptive Predictive Synchronizer [8].

6 References

[1] A. Iyer and D. Marculescu, “Power Efficiency of Voltage Scaling in Multiple Clock, Multiple Voltage Cores", *IEEE/ACM Int. Conf. on Computer Aided Design (ICCAD)*, pp. 379-386, Nov. 2002.

[2] W. J. Dally and J. W. Poulton, “Digital System Engineering”, Cambridge University Press, 1998.

[3] L. Kleeman, A. Cantoni, “Metastable behavior in digital systems”, *IEEE Design and Test of Computers*, pp. 4-19, Dec. 1987.

[4] E.M. Clarke, O. Grumberg and D.A. Peled, “Model Checking”, The MIT Press, 2000.

[5] I. Beer, S. Ben-David, C. Eisner, A. Landver, “RuleBase: an industry-oriented formal verification tool”, *Design Automation Conference*, pp. 665-660 June 1996.

[6] M. Gordon, J. Hurd and K. Slind, “Executing the formal semantics of the Accellera Property Specification Language by mechanised theorem proving,” *CHARME*, LNCS 2860, pp. 200–215, 2003.

[7] T. Kapschitz and R.Ginosar, “Formal Verification of Synchronizers,” CCIT Tech. Rep. 536, EE Dept., Technion, 2005.

[8] U. Frank and R. Ginosar, “A Predictive Synchronizer for Periodic Clock Domains,” *PATMOS*, LNCS 3254, pp. 402–412, 2004.

[9] T. A. Chu, C. K. C. Leung, T. S. Wanuga, "A Design Methodology for Concurrent VLSI Systems", in Proc. of ICCD, 407-410, 1985.

[10] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev, “Complete state encoding based on the theory of regions”, *2nd Int. Symp. Asynchronous Circuits and Systems*, pp. 36-47, March 1996.