Using Model Checking for Synchronizer Verification

Tsachy Kapshitz and Ran Ginosar
Technion, Israel

ACiD-WG, Turku, June 2004
Outline

• The problem
• Structural verification
• 1CD control functional verification
• MCD control functional verification
• MCD data functional verification
Need Formal Approach

• We cannot verify synchronizers merely by logic simulation
  – Continuous (analog) issues are transparent to logic simulation
  – Rare cases (particular relative timing) may evade simulation

• We need a formal method

• Problem:
  – Most model checking tools are synchronous
What is a Synchronizer?

Sender Control

Receiver Control

Synchronization circuit

CLK1

CLK2

Synchronizer
The Goal

- Decompose the synchronizer:
  - Two synchronous blocks
  - Async interconnect
- Apply model checking to the sync blocks
- Verify the async interconnect with something else...
The Method

• Three steps:
  – **Structural: Synchronization circuits**
    • Identify synchronization circuits
    • Verify (e.g. sufficient resolution time)
  – **Structural: Synchronizers**
    • Recognize synchronizers
    • Structurally verify async interconnects
  – **Functional: Verify correctness (model checking)**
    • Decomposed verification—limited to the insides of synchronous clock domains
RuleBase

- Model checker by IBM research
- Luckily, developed next door...
  - www.haifa.il.ibm.com/projects/verification/RB_Homepage/
- Input language: Sugar2 / PSL
- Inputs:
  - The design (Verilog)
  - The environment (e.g. clocks)
  - The rules to be verified
- Also used @Verifier / @Designer from @HDL
  - They incorporate a RuleBase engine
Two-flop synchronizer
Outline

• The problem
• Structural verification
• 1CD control functional verification
• MCD control functional verification
• MCD data functional verification
Functional Control Verification

- Synchronizer protocol:

- STG is
  - Decomposed
  - Translated to Sugar / PSL assertions
Decomposed STG
STG → Assertions

- **Example:**
  - Verify order (L+→L-→R1+ )
    - \( \text{AG} \left( \neg \text{RST} \land \text{rose}(L) \Rightarrow (\text{fell}(L) \text{ before } \neg \text{rose}(R1)) \right) \)
  - Verify state (L=0 on green path)
    - \( \text{AG} \left( \neg \text{RST} \land \text{fell}(L) \Rightarrow (\neg \text{L until fell}(A1)) \right) \)
STG → Assertions

- Assertions generated for each STG node
- assertions generated automatically
STG → Assertions: Problems

- Synchronization protocol specific
- STG should satisfy some properties
  - e.g. Complete State Coding
- Exponential complexity
  - Same as STG→SG
- Requires splitting the protocol into synchronous parts
  - Mitigation: Model asynchronous clocks in RuleBase
Outline

- The problem
- Structural verification
- 1CD control functional verification
- MCD control functional verification
- MCD data functional verification
Clock Modeling in RuleBase

• The Model Checker operates on a sequence of “ticks”

• RuleBase can model flip-flops in two modes:
  – “Level triggered” mode (the default):
    • FFs sample on ticks when clock level=1
  – “Edge triggered” mode:
    • FFs sample on rising edge of the clock

• We use level-triggered mode clocks
Multi-Clock Domain Modeling

• Given two un-related clock domains
  – Mutually asynchronous clocks

• Each tick, we allow their states to proceed or stall non-deterministically

• RuleBase: Achieve non-determinism by assigning a set of values (one of them is selected in random):
  – assign state := \{value1, value2, ..., valueN\};
Non-periodic clocks

• Declaring unrelated clocks:
  
  VAR CLK1, CLK2: 0..1;
  fairness CLK1=1;
  fairness CLK2=1;

• No further assignment into CLK1, CLK2:
  – Each of them may change non-deterministically on every tick

• Fairness:
  – Each clock will change “infinitely” many times
Periodic clocks

• Two clocks with frequency ratio $m:n$ (WLOG $m>n$):
  – Between any two rising edges of CLK2 there are $i$ rising edges of CLK1:
    \[
    \left\lfloor \frac{m}{n} \right\rfloor \leq i \leq \left\lceil \frac{m}{n} \right\rceil
    \]
  – We select $i$ non-deterministically from these two values (and maybe additional ones)
Periodic clocks

- Example: Ratio is 3:2

```plaintext
var c: 0..2;
var CLK, CLK2: 0..1;
assign next(c) := {(c + 1) mod 2), (c + 1) mod 3};
assign CLK1 := c != 0;
assign CLK2 := c = 0;
```

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLK1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLK2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Periodic clocks**

- Example continued—We can also model simultaneous edges:

  ```
  assign CLK1 := if (c != 0) then 1 else {0, 1} endif;
  assign CLK2 := c = 0;
  ```

```
  c  1  2  0  1  0  1  0  1  2

  CLK1
  | ▲ ▲ ▲ ▲ ▲ ▲ ▲ ▲

  CLK2
  . . . . . . . . .
```
Outline

• The problem
• Structural verification
• 1CD control functional verification
• MCD control functional verification
• MCD data functional verification
Data transfer verification

- This is an alternative method of verifying synchronizers, trying to bypass the details of the synchronizer itself.
- Once we identify the following structure,

\[
\text{AG} \left( \neg \text{RST} \land \text{CLK1} \land \text{load} \land \text{DIN}(0)=1 \rightarrow \text{next_event( CLK2 } \land \text{en }) \left( \text{S_BUF}(0)=1 \right) \right)
\]
Data transfer verification (Cont)

- In addition we should verify that:
  - No duplicate: The receiver does not receive data if the sender did not send any
    - AG (\!RST & CLK2 & en -> AX((CLK1 & load) before (CLK2 & en)) )
  - No miss: The receiver eventually receives data that was sent by the sender
    - AG (\!RST & CLK1 & load -> AX((CLK2 & en) before! (CLK1 & load)) )
Summary

• Employ model checking for FV of synchronizers
• Decompose synchronizers into sync components, verify each separately
• Verify async synchronizers
• Verify control and data