Fast Prototyping from Assertions: a Pragmatic Approach

K. Morin-Allory, N. Javaheri, D. Borrione

TIMA Laboratory

October 17, 2013
SoC Design Flow: Protocols or Control Part

System behavior: informal spec

Manually Hardware development

Formal specifications

Hardware verification

Hardware refinement
SoC Design Flow: Protocols or Control Part

- System behavior: informal spec
- Manually Hardware development
- Formal specifications
- Hardware verification

Hardware refinement
SoC Design Flow: Protocols or Control Part

System behavior: informal spec

Automatically correct-by-construction Hardware

SyntHorus2

Formal specifications
Outline

- PSL and the generalized Buffer
- Reactant synthesis
- Annotation and Directed Abstract Syntax Tree (DAST)
- Unsettled direction
- Experimental results
Generalized Buffer

Generalized Buffer

sender#0

sender#i

GenBuf Controller
(automatically generated by Synthorus2)

FIFO

The 4-phase handshake protocol:
The sender side

always ((BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0));
Complete specification of the 4-phase handshake protocol

P1:
always ((not BtoS_ACK_0) and (not StoB_REQ_0) → next! (not BtoS_ACK_0));
P2: always ((BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0));
P3: always (rose(StoB_REQ_0) → not BtoS_ACK_0);
P4: always (not BtoS_ACK_0 or not BtoS_ACK_1);
P5: always (ENQ → BtoS_ACK_0 or BtoS_ACK_1);
P6: always (not ENQ → not BtoS_ACK_0 and not BtoS_ACK_1);
Complete specification of the 4-phase handshake protocol

P1:
always ((not BtoS_ACK_0) and (not StoB_REQ_0) → next! (not BtoS_ACK_0));
P2: always ((BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0));
P3: always (rose(StoB_REQ_0) → not BtoS_ACK_0);
P4: always (not BtoS_ACK_0 or not BtoS_ACK_1);
P5: always (ENQ → BtoS_ACK_0 or BtoS_ACK_1);
P6: always (not ENQ → not BtoS_ACK_0 and not BtoS_ACK_1);
A library of primitive reactants:
- One primitive reactant for each PSL operator
From PSL to reactants

- A library of primitive reactants:
  - One primitive reactant for each PSL operator

- A generic interface and a generic architecture
  - Cond: monitored operand
  - Trigger: start of the next module or generated operands
always (( BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0));
always (( BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0));
always ((BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0))
Formal definition in VLSI’SOC (October 2013, Istanbul)

A down edge means generation

always \((\text{BtoS}_0 \text{ACK}_0 \land \text{StoB}_0 \text{REQ}_0 \rightarrow \text{next!} (\text{BtoS}_0 \text{ACK}_0))\)
Formal definition in VLSI’SOC (october 2013, Istanbul)

Operator always:

\[
\text{always} \rightarrow \text{next! (BtoS\textunderscore ACK\_0)}
\]

\[
\text{always ((BtoS\textunderscore ACK\_0 and StoB\textunderscore REQ\_0 \rightarrow \text{next! (BtoS\textunderscore ACK\_0))}}
\]
Directed Abstract Syntax tree and annotation

Formal definition in VLSI’SOC (october 2013, Istanbul)

Operator imply:
Propagation of a down edge to at least one child

always \(((\text{BtoS}_\text{ACK}_0 \land \text{StoB}_\text{REQ}_0) \rightarrow \text{next!} (\text{BtoS}_\text{ACK}_0))\)
Formal definition in VLSI’SOC (october 2013, Istanbul)

Operator imply:

always ((BtoS_ACK_0 and StoB_REQ_0 → next! (BtoS_ACK_0)))
Formal definition in VLSI’SOC (October 2013, Istanbul)

Propagation of an up edge to both children

always (\((\text{BtoS\_ACK\_0 and StoB\_REQ\_0 \rightarrow next! (\text{BtoS\_ACK\_0}))\))
Formal definition in VLSI’SOC (October 2013, Istanbul)

Operator next:

```
always (((BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0)))
```
always ((BtoS_ACK_0_m and StoB_REQ_0_m) → next! (BtoS_ACK_0_g))
P5: Always ENQ $\rightarrow$ BtoS_ACK_0 or BtoS_ACK_1

Diagram: 
- Always
- ENQ
- or
- BtoS_ACK_0
- BtoS_ACK_1
P5: Always ENQ $\rightarrow$ BtoS_ACK_0 or BtoS_ACK_1

- Signals BtoS_ACK_0 and BtoS_ACK_1 are not annotated.
- Impossible to generate them simultaneously because:
  P4: always (not BtoS_ACK_0 or not BtoS_ACK_1)
Restriction of the graphs to unsettled edge.
Restriction of the graphs to unsettled edge.
Set of properties

Restriction of the graphs to unsettled edge.

Trig5

or

BtoS_ACK_0
BtoS_ACK_1
Restriction of the graphs to unsettled edge.
Restriction of the graphs to unsettled edge.
Computation of signals solved together

- Merge of graphs
Computation of signals solved together

- Merge of graphs

![Diagram](image-url)
Merge of graphs

Computation of strongly connected components
(BtoS_ACK_0, BtoS_ACK_1)
Computation of signals solved together

- Merge of graphs
- Computation of strongly connected components (BtoS_ACK_0, BtoS_ACK_1)
- Extraction of equations

\[
\text{Trig4} \rightarrow \overline{\text{BtoS_ACK}_0} \lor \overline{\text{BtoS_ACK}_1}
\]
\[
\text{Trig5} \rightarrow \overline{\text{BtoS_ACK}_0} \lor \overline{\text{BtoS_ACK}_1}
\]
Trig4 -> BtoS_ACK_0 v BtoS_ACK_1
Trig5 -> BtoS_ACK_0 v BtoS_ACK_1
Trig4 -> BtoS_ACK_0 v BtoS_ACK_1
Trig5 -> BtoS_ACK_0 v BtoS_ACK_1
Trig4 -> BtoS_ACK_0 ∨ BtoS_ACK_1
Trig5 -> BtoS_ACK_0 ∨ BtoS_ACK_1

ABC: http://www.eecs.berkeley.edu/~alanmi/abc/
The whole circuit
The construction flow

P1: always ((not BtoS_ACK.0) and (not StoB_REQ.0) \rightarrow next! (not BtoS_ACK.0));
P2: always ((BtoS_ACK.0 and StoB_REQ.0) \rightarrow next! (BtoS_ACK.0));
P3: always (\text{rose}(StoB_REQ.0) \rightarrow \text{not} BtoS_ACK.0);
P4: always (\text{not} BtoS_ACK.0 \text{ or not} BtoS_ACK.1);
P5: always (ENQ \rightarrow BtoS_ACK.0 \text{ or BtoS_ACK.1});
P6: always (\text{not} ENQ \rightarrow \text{not} BtoS_ACK.0 \text{ and not} BtoS_ACK.1);
P1: always ((not BtoS_ACK_0) and (not StoB_REQ_0) → next! (not BtoS_ACK_0));
P2: always ((BtoS_ACK_0 and StoB_REQ_0) → next! (BtoS_ACK_0));
P3: always (rose(StoB_REQ_0) → not BtoS_ACK_0);
P4: always (not BtoS_ACK_0 or not BtoS_ACK_1);
P5: always (ENQ → BtoS_ACK_0 or BtoS_ACK_1);
P6: always (not ENQ → not BtoS_ACK_0 and not BtoS_ACK_1);
The construction flow

P1: always ((not BtoS_ACK_0) and (not StoS_REQ_0) → next! (not BtoS_ACK_0));
P2: always ((BtoS_ACK_0 and StoS_REQ_0) → next! (BtoS_ACK_0));
P3: always (rise(StoS_REQ_0) → not BtoS_ACK_0);
P4: always (not BtoS_ACK_0 or not BtoS_ACK_1);
P5: always (ENQ → BtoS_ACK_0 or BtoS_ACK_1);
P6: always (not ENQ → not BtoS_ACK_0 and not BtoS_ACK_1);
The construction flow

P1: always ((not BtoS.ACK.0) and (not StoB.REQ.0) → next! (not BtoS.ACK.0));
P2: always ((BtoS.ACK.0 and StoB.REQ.0) → next! (BtoS.ACK.0));
P3: always (rise(S-to-B.REQ.0) → not BtoS.ACK.0);
P4: always (not BtoS.ACK.0 or not BtoS.ACK.1);
P5: always (ENQ → BtoS.ACK.0 or BtoS.ACK.1);
P6: always (not ENQ → not BtoS.ACK.0 and not BtoS.ACK.1);
The construction flow

P1: always ((not BtoS.ACK_0) and (not StoB.REQ_0) — next! (not BtoS.ACK_0));
P2: always ((BtoS.ACK_0 and StoB.REQ_0) — next! (BtoS.ACK_0));
P3: always (rose(StoB.REQ_0) — not BtoS.ACK_0);
P4: always (not BtoS.ACK_0 or not BtoS.ACK_1);
P5: always (ENQ — BtoS.ACK_0 or BtoS.ACK_1);
P6: always (not ENQ — not BtoS.ACK_0 and not BtoS.ACK_1);
Results

Hw. generation time (s)

Results

Total No. of Gates

Conclusion: A new design flow?

Is it Possible? Yes

System behavior: informal spec

Formal specifications

Automatically correct-by-construction Hardware
Conclusion: A new design flow?

Is it Possible? Yes
Is it worth it? Well ...
Fast prototyping from assertions: what good is it for?

- Defining and verifying the specifications:
  - Generation of waveforms
  - Generation of theorems to prove the consistency and the completeness of the specification
- Generation of a golden model (only if the specification was complete)
- Generation of a simplified environment for on line testing of a design
  - simulation
  - prototyping
Defining and verifying the specifications:
- Generation of waveforms
- Generation of theorems to prove the consistency and the completeness of the specification
- Generation of a golden model (only if the specification was complete)
- Generation of a simplified environment for on line testing of a design
  - simulation
  - prototyping
Defining and verifying the specifications:
- Generation of waveforms
- Generation of theorems to prove the consistency and the completeness of the specification

Generation of a golden model (only if the specification was complete)

Generation of a simplified environment for on line testing of a design
- simulation
- prototyping
Defining and verifying the specifications:
- Generation of waveforms
- Generation of theorems to prove the consistency and the completeness of the specification
- Generation of a golden model (only if the specification was complete)
- Generation of a simplified environment for on line testing of a design
  - simulation
  - prototyping
Future Works

- Combine with a model checker to verify the specification coherency and consistency
- Apply optimizations on the design
- Work on bit vectors and integers
- Define guidelines for the PSL subset for synthesis