A Review of
the Self-Timed Circuit Verification Framework

Cuong Chau
ckuong@cs.utexas.edu
Department of Computer Science
The University of Texas at Austin

ACL2 Seminar Talk

November 30, 2018
Outline

1. Verification Flow

2. Self-Timed Circuits with Deterministic Outputs

3. Self-Timed Circuits with Non-Deterministic Outputs

4. Conclusion
Outline

1. Verification Flow
2. Self-Timed Circuits with Deterministic Outputs
3. Self-Timed Circuits with Non-Deterministic Outputs
4. Conclusion
Verification Flow

Hierarchical reasoning & induction

- Single-step-update properties
  - Value and state lemmas
  - Induction
  - run

- Multi-step input-output relationship
  - Induction

- Multi-step state lemma
  - Functional correctness

C. Chau (UT Austin)
Value and State Lemmas

**Value lemma:** characterize the module’s outputs
\[ se(module-name, inputs, st, netlist) = outputs(inputs, st) \]

**State lemma:** characterize the module’s next state
\[ de(module-name, inputs, st, netlist) = step(inputs, st) \]

Functions *outputs* and *step* are **hierarchically** defined as **symbolic, four-valued expressions** that specify the module’s outputs and next state, respectively.
Multi-Step State Lemma

Characterize the module’s final state after an $n$-step execution.

$$de-n(module-name, inputs-seq, st, netlist, n) = run(inputs-seq, st, n)$$
Multi-Step State Lemma

Characterize the module’s final state after an \( n \)-step execution.

\[ de-n(module-name, inputs-seq, st, netlist, n) = run(inputs-seq, st, n) \]

\[ run(inputs-seq, st, n) := \]

\[ \text{if } (n \leq 0) \text{ st} \]

\[ \text{else} \]

\[ run(rest(inputs-seq), \]

\[ step(first(inputs-seq), st), \]

\[ n - 1) \]
Single-Step-Update Property

Specify the input-output relationship after one execution step.

Introduce an extraction function for each self-timed module, $extract(st)$, that returns a sequence of values computed from valid data residing in state $st$.

Applying $extract$ to $step$ will compute the one-step update on the output sequence given the current inputs and current state. Note that $extract$ and $step$ are defined hierarchically.
Single-Step-Update Property

Specify the input-output relationship after **one execution step**.

Introduce an **extraction function** for each self-timed module, \( \text{extract}(st) \), that returns a sequence of values computed from **valid data residing in state** \( st \).

Applying \( \text{extract} \) to \( \text{step} \) will compute the **one-step update** on the output sequence given the current inputs and current state. Note that \( \text{extract} \) and \( \text{step} \) are defined **hierarchically**.

**Single-step-update property:**

\[
\text{extract}(\text{step}(\text{inputs}, st)) = \text{extracted-step}(\text{inputs}, st)
\]

where **extracted-step** is the specification for the one-step update on the output sequence.
Single-Step-Update Property

**Example:** Let \( \text{in-act} \) and \( \text{out-act} \) denote the **communication signals** at the input and output ports respectively (Assume that the corresponding module has one input and one output ports).

\[
\text{extracted-step}(\text{inputs, st}) :=
\begin{cases}
\text{extract}(\text{st}), & \text{if } \text{in-act} = \text{nil} \land \text{out-act} = \text{nil} \\
[\text{op}(\text{inputs.data})] \, \text{++} \, \text{extract}(\text{st}), & \text{if } \text{in-act} = \text{t} \land \text{out-act} = \text{nil} \\
\text{remove-last}(\text{extract}(\text{st})), & \text{if } \text{in-act} = \text{nil} \land \text{out-act} = \text{t} \\
[\text{op}(\text{inputs.data})] \, \text{++} \, \text{remove-last}(\text{extract}(\text{st})), & \text{otherwise}
\end{cases}
\]

where

- \( \text{++} \) is the concatenation operator;
- \( \text{remove-last}(l) \) returns list \( l \) except for its last element; and
- \( \text{op} \) is the **functional specification** for the module.
Multi-Step Input-Output Relationship

We verify the functional correctness of self-timed circuits in terms of the relationship between their input and output sequences.

Our formalization considers a general case where:

- the initial state may contain some valid data; and
- there can be some valid data remaining in the final state.

**Example:**

\[
\text{extract}(\text{run}(\text{inputs-seq, st, n})) ++ \text{out-seq} = \\
\text{op-map}(\text{in-seq}) ++ \text{extract}(\text{st})
\]
We verify the **functional correctness** of self-timed circuits in terms of the relationship between their input and output sequences.

Our formalization considers a general case where:
- the initial state may contain some valid data; and
- there can be some valid data remaining in the final state.

**Example:**

\[
\text{extract} \left( \text{run}(\text{inputs-seq}, \text{st}, n) \right) ++ \text{out-seq} = \\
\text{op-map}(\text{in-seq}) ++ \text{extract}(\text{st})
\]

The **functional correctness theorem** is a direct corollary of the multi-step input-output relationship that is stated in terms of the \textit{de-n} function, while that relationship is formalized in terms of the \textit{run} function.
Outline

1 Verification Flow

2 Self-Timed Circuits with Deterministic Outputs

3 Self-Timed Circuits with Non-Deterministic Outputs

4 Conclusion
A FIFO Queue of Two Links
Outline

1 Verification Flow

2 Self-Timed Circuits with Deterministic Outputs

3 Self-Timed Circuits with Non-Deterministic Outputs

4 Conclusion
Arbitrated merge is a well-known self-timed circuit model that provides mutually exclusive access to a shared resource.

Produce non-deterministic output sequences due to arbitrary arrival times of requests.

We formalize an arbitrated merge joint that provides mutually exclusive access to its output link from its two input links on a first-come-first-served basis.

---

Circuits Performing Arbitrated Merges

interl

\[ \text{in}_0 \rightarrow Q'_{20a} \rightarrow A \rightarrow \text{out} \]
\[ \text{in}_1 \rightarrow Q'_{20b} \rightarrow \text{arbitrated merge} \]

interl-gcd

\[ \text{in}_0 \rightarrow 2n \rightarrow \text{interl} \rightarrow L \rightarrow \text{gcd} \rightarrow n \rightarrow \text{out} \]
\[ \text{in}_1 \rightarrow 2n \rightarrow \text{gcd} \rightarrow n \rightarrow \text{out} \]

comp-interl

\[ \text{in}_0 \rightarrow \text{interl}_0 \rightarrow L_0 \rightarrow \text{out} \]
\[ \text{in}_1 \rightarrow \text{interl}_0 \rightarrow L_0 \rightarrow \text{out} \]
\[ \text{in}_2 \rightarrow \text{interl}_1 \rightarrow L_1 \rightarrow \text{out} \]
\[ \text{in}_3 \rightarrow \text{interl}_1 \rightarrow L_1 \rightarrow \text{out} \]
Hierarchical reasoning & induction

- Single-step-update properties
- Value and state lemmas

Induction

- Multi-step input-output relationship
- Multi-step state lemma

Induction

Run

Functional correctness
Verification Flow

Hierarchical reasoning & induction

Single-step-update properties

Induction

Multi-step input-output relationship

Value and state lemmas

Induction

Multi-step state lemma

Functional correctness
Define **two extraction functions** for each arbitrated merge, one for each input stream.

**Single-step-update properties:**

\[
\begin{align*}
\text{extract}_0(\text{step}(\text{inputs}, \text{st})) &= \text{extracted}_0\text{-step}(\text{inputs}, \text{st}) \\
\text{extract}_1(\text{step}(\text{inputs}, \text{st})) &= \text{extracted}_1\text{-step}(\text{inputs}, \text{st})
\end{align*}
\]
The multi-step input-output relationship is established using the membership relation \((\in\)) and the interleaving operation \((\otimes)\).

\[ in_0 \rightarrow Q'_{20a} \rightarrow A \rightarrow out \]

\[ in_1 \rightarrow Q'_{20b} \]

\(\text{interl} extract_0\) and \(\text{interl} extract_1\) extract valid data from two complex links \(Q'_{20a}\) and \(Q'_{20b}\), respectively.

\textbf{let}\; st_f := \text{interl} run(in\text{-seq}, st, n),

\[ \forall x \in (\text{interl} extract_0(st_f) \otimes \text{interl} extract_1(st_f)). \]

\[ (x ++ \text{out-seq}) \in \left((in_0\text{-seq} ++ \text{interl} extract_0(st)) \otimes (in_1\text{-seq} ++ \text{interl} extract_1(st))\right) \]
Outline

1. Verification Flow
2. Self-Timed Circuits with Deterministic Outputs
3. Self-Timed Circuits with Non-Deterministic Outputs
4. Conclusion
Conclusion

Reviewed our ACL2 verification framework for self-timed circuit designs.

Illustrated the framework through two examples.

- a self-timed circuit with **deterministic** outputs: a FIFO queue of two links; and
- a self-timed circuit with **non-deterministic** outputs: a circuit performing arbitrated merges.
Questions?