Toward the Formal Verification of HILECOP

Formalization and Implementation of Synchronously Executed Petri Nets

PhD student:
Vincent Iampietro

PhD supervisors:
David Andreu, David Delahaye

PNSE, June 2020
Context and Research Question
Context: The HILECOP Methodology

Overview

- **Purpose:** design and production of critical digital systems.
- **Developed by:** the CAMIN (former DEMAR) team (INRIA).
- **Currently applied:** to build implantable medical devices [Andreu et al., 2009].

Research Question

How can we verify that the translation from ② to ③ is behavior preserving?
Translation (Compilation) Verification

Verification of Programming Language Compilers
A lot of work in the domain: CompCert [Leroy, 2009], [Chlipala, 2010], [Dave, 2003]

Verification Task Procedure

1. Model the source language syntax and semantics:
   ⇒ in HILECOP, abstract source formalism (particular Petri Nets).

2. Model the target language syntax and semantics:
   ⇒ in HILECOP, a Hardware Description Language (HDL).

3. Implement the translation from source to target programs.

4. Prove that the translation is behavior preserving.

Thesis Goal

▶ Prove behavior preservation from ② to ③ (cf. HILECOP fig.).
▶ Verify the proof with the Coq proof assistant.
HILECOP High Level Formalism
HILECOP High Level Formalism

Component 1

behavior

Component 2

behavior
Implementation Model

Context and Research Question  HILECOP High Level Formalism  Implementation  Conclusion 4 / 20
The SITPN Formalism

Synchronously executed Interpreted Time Petri Nets (with priorities).

Overview

▶ Combination of known PN classes
▶ Synchronous execution i.e, transition firing is a synchronous process.

PN Classes

▶ Generalized, extended.
▶ Interpreted: conditions, functions, actions.
▶ Time: intervals, counters, and imperative policy.
The SITPN Formalism

Synchronously executed Interpreted Time Petri Nets (with priorities).

Overview

- Combination of known PN classes
- Synchronous execution i.e, transition firing is a synchronous process.

PN Classes

- Generalized, extended.
- Interpreted: conditions, functions, actions.
- Time: intervals, counters, and imperative policy.
Synchronous Execution

- Due to synchronous implementation on FPGA.
- Falling edge (1):
  - Updates condition values.
  - Updates time counters.
  - Activates actions.
  - Elects transitions to be fired.
- Rising edge (2):
  - Updates the marking.
  - Computes reset orders/blocked time counters.
  - Executes functions.
- Conflicts:
  - Structurally solved.
  - Solved with priorities.
Synchronous Execution

- Due to synchronous implementation on FPGA.

- **Falling edge (1):**
  - Updates condition values.
  - Updates time counters.
  - Activates actions.
  - Elects transitions to be fired.

- **Rising edge (2):**
  - Updates the marking.
  - Computes reset orders/blocked time counters.
  - Executes functions.

- **Conflicts:**
  - Structurally solved.
  - Solved with priorities.
SITPN Structure Formal Definition

An SITPN is a record

- $P = \{P_0, \ldots, P_i\}$, a set of places.
- $T = \{T_0, \ldots, T_j\}$, a set of transitions.
- $A = \{a_0, \ldots, a_k\}$, a set of actions.
- $C = \{c_0, \ldots, c_l\}$, a set of conditions.
- $F = \{f_0, \ldots, f_m\}$, a set of functions.
- $\text{pre} \in P \rightarrow T \rightarrow (\mathbb{N} \times \{\text{basic, inhib, test}\})$.
- $\text{post} \in T \rightarrow P \rightarrow \mathbb{N}$.
- $A \in P \rightarrow A \rightarrow \mathbb{B}$.
- $F \in T \rightarrow F \rightarrow \mathbb{B}$.
- $C \in T \rightarrow C \rightarrow \{-1, 0, 1\}$.
- $I_s \in T \rightarrow \mathbb{I}^+ \ (\mathbb{I}^+ = [a, b] \text{ where } a \in \mathbb{N}^* \text{ and } b \in \mathbb{N}^* \sqcup \{\infty\})$.
- $M_0 \in P \rightarrow \mathbb{N}$.
- $\succ \subseteq (T \times T)$, the firing priority strict order.
An SITPN state is a record

- \( \text{Fired} \subseteq T \).
- \( M \in P \rightarrow \mathbb{N} \).
- \( T_c \in T_i \rightarrow \mathbb{N} \sqcup \{\psi\} \) where \( T_i = \text{dom}(I_s) \).
- \( \text{reset}_t \in T_i \rightarrow \mathbb{B} \).
- \( \text{ex} \in A \sqcup F \rightarrow \mathbb{B} \).
- \( \text{cond} \in C \rightarrow \mathbb{B} \).
SITPN Formal Semantics

The SITPN semantics is a state-transition system.

State Transition Relation

Rules on rising edge.

\[ E, \tau \vdash s \uparrow \Rightarrow s' \]

Rules on falling edge.

\[ E, \tau \vdash s \downarrow \Rightarrow s' \]

where:

- \( E \in \mathbb{N} \rightarrow C \rightarrow \mathbb{B} \) is the environment function.
- \( \tau \), the current count of clock cycles.
- \( s, s' \), two states of a given SITPN.
A Falling Edge Rule Example

\[ \forall t \in \text{firable}(s'), \ t \in \text{sens}(s.M - \sum_{t_i \in \text{Pr}(t)} \text{pre}(t_i)) \Rightarrow t \in s'.\text{Fired} \]

\[ E, \tau \vdash s \Downarrow s' \]

where \( \text{Pr}(t) = \{ t_i \mid t_i \succ t \land t_i \in \text{firable}(s') \} \)

**Firable Transition**

A transition is firable at a given state \( s \) if:

- It is enabled by the current marking.
- Its time counter is in the firing time interval.
- All associated conditions are true.
Implementation
The Coq Proof Assistant

- Interactive theorem proving.
- Language developed by the INRIA (based on OCaml) [The Coq Development Team, 2020].
- Based on the calculus of inductive constructions and the Curry-Howard correspondence.
- A part dedicated to programs, program specifications, theorems, lemmas...
- A part dedicated to proofs (tactics).
The SITPN structure is a record

- $P = \{P_0, \ldots, P_i\}$
- $T = \{T_0, \ldots, T_j\}$
- $A = \{a_0, \ldots, a_k\}$
- $C = \{c_0, \ldots, c_l\}$
- $F = \{f_0, \ldots, f_m\}$
- $\text{pre} \in P \rightarrow T \rightarrow (\mathbb{N} \times \{\text{basic, inhib, test}\})$. 
- $\text{post} \in T \rightarrow P \rightarrow \mathbb{N}$.
- $A \in P \rightarrow A \rightarrow \mathbb{B}$.
- $F \in T \rightarrow F \rightarrow \mathbb{B}$.
- $C \in T \rightarrow C \rightarrow \{-1, 0, 1\}$.
- $I_s \in T \rightarrow \mathbb{I}^+$. 
- $M_0 \in P \rightarrow \mathbb{N}$.
- $\succeq \subseteq (T \times T)$.
SITPN State Implementation

Record SitpnState : Set := mk_SitpnState {  
  fired : list Trans;
  marking : list (Place * nat);
  time_counters : list (Trans * nat);
  reset : list (Trans * bool);
  exec_a : list (Action * bool);
  exec_f : list (IFunction * bool);
  cond_values : list (Condition * bool);
}.

An SITPN state is a record

- $\text{Fired} \subseteq T$.
- $M \in P \rightarrow \mathbb{N}$.
- $T_c \in T_i \rightarrow \mathbb{N} \sqcup \{\psi\}$.
- $\text{reset}_t \in T_i \rightarrow \mathbb{B}$.
- $ex \in A \sqcup F \rightarrow \mathbb{B}$.
- $cond \in C \rightarrow \mathbb{B}$.
SITPN State Transition Relation Implementation

Rules on rising edge.

\[ E, \tau \vdash s \xrightarrow{\uparrow} s' \]

Rules on falling edge.

\[ E, \tau \vdash s \xrightarrow{\downarrow} s' \]

Inductive \texttt{SitpnStateTransition} (\texttt{sitpn : Sitpn})

\[(E : \text{nat} \rightarrow \text{Condition} \rightarrow \text{bool})\]

\[(\tau : \text{nat}) (s s' : \text{SitpnState}) : \text{Clock} \rightarrow \text{Prop} := \]

\texttt{SitpnStateTransition\_rising\_edge}:

(* Ensures the consistency of sitpn, s and s'. *)

\texttt{IsWellDefinedSitpn sitpn} \rightarrow
\texttt{IsWellDefinedSitpnState sitpn s} \rightarrow
\texttt{IsWellDefinedSitpnState sitpn s'} \rightarrow

(* Rules on rising edge *)

...

(** Conclusion **)\n
\texttt{SitpnStateTransition sitpn E \tau s s' \uparrow}

\texttt{SitpnStateTransition\_falling\_edge}:

(* Ensures the consistency of sitpn, s and s'. *)

\texttt{SitpnStateTransition sitpn E \tau s s' \downarrow}.

(* Rules on falling edge*)

...
Falling Edge Rule Implementation Example

\[ \forall t \in firable(s'), \; t \in sens(s.M - \sum_{t_i \in Pr(t)} pre(t_i)) \Rightarrow t \in s'.Fired \]

\[ E, \tau \vdash s \Downarrow s' \]

where \( Pr(t) = \{ t_i \mid t_i \succ t \land t_i \in firable(s') \} \)

\[ \cdots \]

\((\forall \text{ pgroup : list Trans}) (t : \text{ Trans}) (pr : \text{ list Trans}) (resm : \text{ list (Place \star nat)}),\]

\( \text{IsFirable sitpn s' t} \to \)

\( \text{In pgroup sitpn.} (\text{priority_groups}) \to \)
\( \text{In t pgroup } \to \)
\( \text{IsPrioritySet s' pgroup t pr } \to \)
\( \text{IsResidualMarking sitpn s pr resm } \to \)
\( \text{IsSensitized sitpn resm t } \to \)

\( \text{In t s'.} (\text{fired}) ) \to \)

\( \cdots \)

\( \text{SitpnStateTransition sitpn E } \tau s s' \Downarrow \)
Validate Our Implementation

Why?
The proof of behavior preservation will be based on our Coq implementation of the SITPN semantics.

Validation Technics

▶ Quality audit of the implementation by an expert.
▶ Write a functional program that implements the execution semantics (i.e, a token player):
  • Prove that the program complies with the semantics (soundness and completeness).
  • Run tests on the program.
The Token Player Algorithm

\[
\text{cycle}(\text{sitpn}, E, \tau, s)
\]

\[
s' \leftarrow \text{rising}(\text{sitpn}, s); s'' \leftarrow \text{falling}(\text{sitpn}, E, \tau, s'); \text{return} (s', s'')
\]

\[
\text{rising}(\text{sitpn}, s)
\]

\[
s' \leftarrow s
\]

\[
\text{transient\_marking} \leftarrow \text{consumeTokens}(\text{sitpn}, s)
\]

\[
s'.T_c, s'.\text{reset} \leftarrow \text{getBlockedAndResetOrds}(\text{sitpn}, s, \text{transient\_marking})
\]

\[
s'.M \leftarrow \text{produceTokens}(\text{sitpn}, s, \text{transient\_marking})
\]

\[
s'.\text{ex} \leftarrow \text{getFunctionStates}(\text{sitpn}, s)
\]

\[
\text{return} s'
\]

\[
\text{falling}(\text{sitpn}, E, \tau, s)
\]

\[
s' \leftarrow s
\]

\[
s'.\text{cond} \leftarrow \text{getConditionValues}(\text{sitpn}, E, \tau)
\]

\[
s'.\text{ex} \leftarrow \text{getActionStates}(\text{sitpn}, s)
\]

\[
s'.T_c \leftarrow \text{updateTimeIntervals}(\text{sitpn}, s)
\]

\[
s'.\text{Fired} \leftarrow \text{electToBeFired}(\text{sitpn}, s')
\]

\[
\text{return} s'
\]
Soundness and Completeness Theorems

Soundness Theorem
\[ \forall \text{sitpn}, E, \tau, s, s', \; \text{cycle}(\text{sitpn}, E, \tau, s) = \text{Some } s' \Rightarrow E, \tau \vdash \text{sitpn}, s \uparrow, \downarrow \rightarrow s' \]

Completeness Theorem
\[ \forall \text{sitpn}, E, \tau, s, s', \; E, \tau \vdash \text{sitpn}, s \uparrow, \downarrow \rightarrow s' \Rightarrow \text{cycle}(\text{sitpn}, E, \tau, s) = \text{Some } s' \]
Conclusion
Conclusion

▶ Thesis Goal: formally verify a translation pass in the HILECOP methodology.
▶ First step: express the semantics of SITPNs (HILECOP high level formalism).

Contributions

▶ Implementation of SITPNs and firing semantics in Coq.
▶ Verified token player.
▶ *Formalization and implementation of a synchronous semantics for VHDL.*
▶ *Implementation of the translation from SITPNs to VHDL designs.*

Perspectives

▶ Prove the behavior preservation theorem!
▶ SITPN formalism at its full potential:
  • Macroplaces (exception handling structures).
  • Multi-clock domains (Globally Asynchronous Locally Synchronous systems).
SITPN Implementation GitHub Repository

https://github.com/viampietro/sitpns


