Modeling and Verifying Asynchronous Circuits Using the DE System

Cuong Chau

ckcuong@cs.utexas.edu

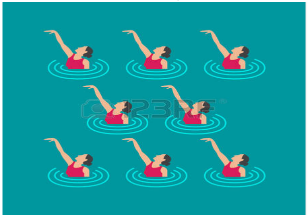
Department of Computer Science
The University of Texas at Austin

Ph.D. Dissertation Proposal

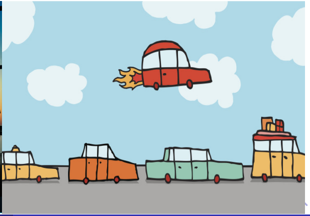
April 9, 2018

Introduction

Synchronous circuits (or clocked circuits): changes in the state of storage elements are synchronized by **a global clock signal**.



Asynchronous circuits (or self-timed circuits): no global clock signal. The communications between storage elements are performed via **local communication protocols**.



Motivation

Most efforts in verifying self-timed circuit implementations concern **circuit-level timing properties**.

Electrical-level timing analysis is conducted to assure that signal propagation of ready signals is always slower than data propagation so that **data are valid when transferred**.

Motivation

Most efforts in verifying self-timed circuit implementations concern **circuit-level timing properties**.

Electrical-level timing analysis is conducted to assure that signal propagation of ready signals is always slower than data propagation so that **data are valid when transferred**.

Most verification methods for self-timed circuits have concentrated on **small-size** circuits.

Motivation

Most efforts in verifying self-timed circuit implementations concern **circuit-level timing properties**.

> Electrical-level timing analysis is conducted to assure that signal propagation of ready signals is always slower than data propagation so that **data are valid when transferred**.

Most verification methods for self-timed circuits have concentrated on **small-size** circuits.

Scalable methods for self-timed system verification are highly desirable.

Motivation

Most efforts in verifying self-timed circuit implementations concern **circuit-level timing properties**.

> Electrical-level timing analysis is conducted to assure that signal propagation of ready signals is always slower than data propagation so that **data are valid when transferred**.

Most verification methods for self-timed circuits have concentrated on **small-size** circuits.

Scalable methods for self-timed system verification are highly desirable.

We are not aware of any scalable formal methods for validating functional properties of self-timed systems.

Goals and Impact

Goals:

- Develop scalable methods for reasoning about the functional correctness of self-timed circuits and systems, while **abstracting away circuit-level timing constraints**.
- Implement those methods using the ACL2 theorem proving system, providing a useful automated framework with associated libraries to support the mechanical analysis of general-purpose, self-timed circuit designs.

Goals and Impact

Goals:

- Develop scalable methods for reasoning about the functional correctness of self-timed circuits and systems, while **abstracting away circuit-level timing constraints**.
- Implement those methods using the ACL2 theorem proving system, providing a useful automated framework with associated libraries to support the mechanical analysis of general-purpose, self-timed circuit designs.

Impact: If successful, this project will:

- advance the state-of-the-art in self-timed circuit specification and verification, and provide a means to support building **reliable complex hardware systems** using the self-timed paradigm; and thus,
- support a computing paradigm where **systems can proceed at their best rate and no longer require clock signals**.

Approach

Extend the DE-based, synchronous-style verification system [Hunt:2000] to one that is capable of analyzing self-timed system models.

Approach

Extend the DE-based, synchronous-style verification system [Hunt:2000] to one that is capable of analyzing self-timed system models.

Apply the link-joint model introduced by Roncken et al. [Roncken et al.:2015] to modeling self-timed circuit designs.

Approach

Extend the DE-based, synchronous-style verification system [Hunt:2000] to one that is capable of analyzing self-timed system models.

Apply the link-joint model introduced by Roncken et al. [Roncken et al.:2015] to modeling self-timed circuit designs.

Develop a hierarchical reasoning approach that is amenable to verifying correctness of large, non-deterministic systems without a large growth of the time complexity.

Approach

Extend the DE-based, synchronous-style verification system [Hunt:2000] to one that is capable of analyzing self-timed system models.

Apply the link-joint model introduced by Roncken et al. [Roncken et al.:2015] to modeling self-timed circuit designs.

Develop a hierarchical reasoning approach that is amenable to verifying correctness of large, non-deterministic systems without a large growth of the time complexity.

- Avoid exploring the operations **internal to a verified submodule** as well as their interleavings.
- The **input-output relationship** of a verified submodule is determined based on the communication signals at the submodule's input and output ports, while **abstracting away all execution paths internal to that submodule**.

What Has Been Done?

- Extended the DE system to modeling self-timed circuit designs.
  - Extended the DE primitive database with a new link-control primitive that coordinates the means to update the state of a (storage) link.
  - Formally specified several self-timed circuit models using the extended DE system.

What Has Been Done?

- Extended the DE system to modeling self-timed circuit designs.
  - Extended the DE primitive database with a new link-control primitive that coordinates the means to update the state of a (storage) link.
  - Formally specified several self-timed circuit models using the extended DE system.
- Developed a hierarchical verification approach that scales well even as circuit size increases.

  Implemented strategies for reasoning with non-deterministic circuit behavior efficiently.

What Has Been Done?

- Extended the DE system to modeling self-timed circuit designs.
  - Extended the DE primitive database with a new link-control primitive that coordinates the means to update the state of a (storage) link.
  - Formally specified several self-timed circuit models using the extended DE system.
- Developed a hierarchical verification approach that scales well even as circuit size increases.

  Implemented strategies for reasoning with non-deterministic circuit behavior efficiently.

- Successfully applied our verification approach to several self-timed circuit models.

Future Work

Goal: Demonstrate the effectiveness of our compositional, mechanized methodology for scalable formal verification of functional properties of self-timed circuit designs.

Future Work

Goal: Demonstrate the effectiveness of our compositional, mechanized methodology for scalable formal verification of functional properties of self-timed circuit designs.

Proposed tasks:

- Enhance the effectiveness of our framework by increasing automation through the further introduction of **proof idioms** using **macros**.

Future Work

Goal: Demonstrate the effectiveness of our compositional, mechanized methodology for scalable formal verification of functional properties of self-timed circuit designs.

Proposed tasks:

- Enhance the effectiveness of our framework by increasing automation through the further introduction of **proof idioms** using **macros**.
- Verify self-timed circuit models performing arbitrated merge operations that grant mutually exclusive access to a shared resource on a **first-come-first-served** (FCFS) basis.

Future Work

Goal: Demonstrate the effectiveness of our compositional, mechanized methodology for scalable formal verification of functional properties of self-timed circuit designs.

Proposed tasks:

- Enhance the effectiveness of our framework by increasing automation through the further introduction of **proof idioms** using **macros**.
- Verify self-timed circuit models performing arbitrated merge operations that grant mutually exclusive access to a shared resource on a **first-come-first-served** (FCFS) basis.
- Verify a self-timed **serial adder** model without imposing the design restrictions inherent in our previous work [Chau et al.:2017].

Future Work

Goal: Demonstrate the effectiveness of our compositional, mechanized methodology for scalable formal verification of functional properties of self-timed circuit designs.

Proposed tasks:

- Enhance the effectiveness of our framework by increasing automation through the further introduction of **proof idioms** using **macros**.
- Verify self-timed circuit models performing arbitrated merge operations that grant mutually exclusive access to a shared resource on a **first-come-first-served** (FCFS) basis.
- Verify a self-timed **serial adder** model without imposing the design restrictions inherent in our previous work [Chau et al.:2017].
- Demonstrate compositionality by certifying that the functionality of *gcd* is preserved when replacing its **combinational ripple-carry-adder** sub-circuit with a functionally-equivalent, **self-timed serial adder**.

Outline

1 The DE System

2 Modeling and Verification Approach

3 Case Studies

4 Conclusions

Outline

1 The DE System

2 Modeling and Verification Approach

3 Case Studies

4 Conclusions

The DE System

DE is a formal occurrence-oriented hardware description language developed in ACL2 for describing Mealy machines [Hunt:2000].

The DE System

DE is a formal occurrence-oriented hardware description language developed in ACL2 for describing Mealy machines [Hunt:2000].

The semantics of DE is given by a simulator that computes the **outputs** and **next state** for a module from the module's **current inputs** and **current state**.

The DE System

DE is a formal occurrence-oriented hardware description language developed in ACL2 for describing Mealy machines [Hunt:2000].

The semantics of DE is given by a simulator that computes the **outputs** and **next state** for a module from the module's **current inputs** and **current state**.

The DE system has previously been used to model and verify hierarchical synchronous circuits.

- The DE simulator is used repeatedly to evaluate a circuit netlist description at **each global clock "tick"**.

The DE System

DE is a formal occurrence-oriented hardware description language developed in ACL2 for describing Mealy machines [Hunt:2000].

The semantics of DE is given by a simulator that computes the **outputs** and **next state** for a module from the module's **current inputs** and **current state**.

The DE system has previously been used to model and verify hierarchical synchronous circuits.

- The DE simulator is used repeatedly to evaluate a circuit netlist description at **each global clock "tick"**.
- Prove the following two lemmas **hierarchically** for each module: a value lemma specifying the module's outputs and a state lemma specifying the module's next state.

The DE System

DE is a formal occurrence-oriented hardware description language developed in ACL2 for describing Mealy machines [Hunt:2000].

The semantics of DE is given by a simulator that computes the **outputs** and **next state** for a module from the module's **current inputs** and **current state**.

The DE system has previously been used to model and verify hierarchical synchronous circuits.

- The DE simulator is used repeatedly to evaluate a circuit netlist description at **each global clock "tick"**.
- Prove the following two lemmas **hierarchically** for each module: a value lemma specifying the module's outputs and a state lemma specifying the module's next state.
- The value and state lemmas of **composite modules** are proved by automatic application of those lemmas of their submodules, **without the need to dig into any details about the submodules**.

The DE System

In our self-timed modeling, the DE simulator is called upon to carry out its function any time any primary input or internal state changes value.

Allow the design to proceed at its own rate moderated by **oracle** values — extra input values modeling non-determinacy — that can cause logic to **delay an arbitrary amount**.

The DE System

In our self-timed modeling, the DE simulator is called upon to carry out its function any time any primary input or internal state changes value.

> Allow the design to proceed at its own rate moderated by **oracle** values — extra input values modeling non-determinacy — that can cause logic to **delay an arbitrary amount**.

Extend the DE primitive database with a link-control primitive that models the **validity of data** stored in a communication link.

Outline

1 The DE System

2 Modeling and Verification Approach

3 Case Studies

4 Conclusions

The Link-Joint Model

We model self-timed systems as **finite state machines** (FSMs) representing networks of communication links and computation joints.

Links communicate with each other locally via joints using the **link-joint model**.

The Link-Joint Model

We model self-timed systems as **finite state machines** (FSMs) representing networks of communication links and computation joints.

Links communicate with each other locally via joints using the **link-joint model**.

- Links are communication channels in which **data** are stored along with **a full/empty signal**.
- Joints are handshake components that implement **data operations** and **flow control**.
- Links are connected via joints, and joints are connected via links. A joint can have several input and output links connected to it, while a link connects exactly to one input and one output joint.

The Link-Joint Model

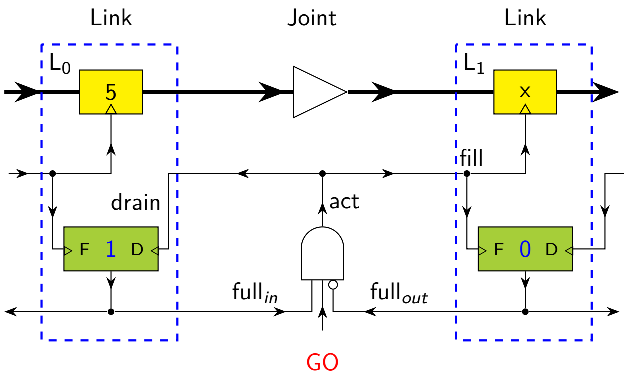
We model self-timed systems as **finite state machines** (FSMs) representing networks of communication links and computation joints.

Links communicate with each other locally via joints using the **link-joint model**.

- Links are communication channels in which **data** are stored along with **a full/empty signal**.
- Joints are handshake components that implement **data operations** and **flow control**.
- Links are connected via joints, and joints are connected via links. A joint can have several input and output links connected to it, while a link connects exactly to one input and one output joint.

Necessary conditions for a **joint-action** to fire: all input and output links of that action are **full** and **empty**, respectively.

The Link-Joint Model

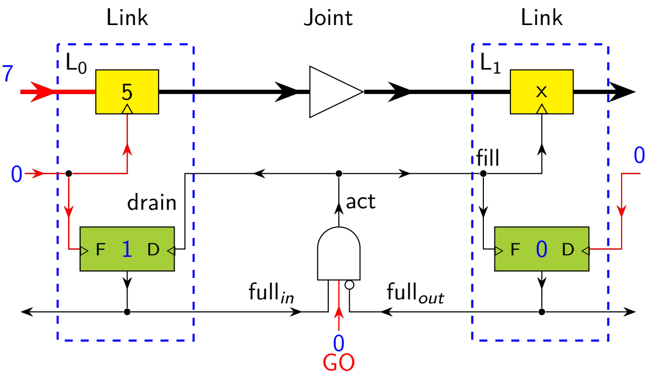


The green boxes represent the instances of the link-control primitive that is added to the DE primitive database.

When a joint acts, three tasks will be executed in parallel:

- transfer data computed from the input links to the output links;
- fill a subset of the output links, leaving them **full**;
- drain a subset of the input links, leaving them **empty**.

The Link-Joint Model

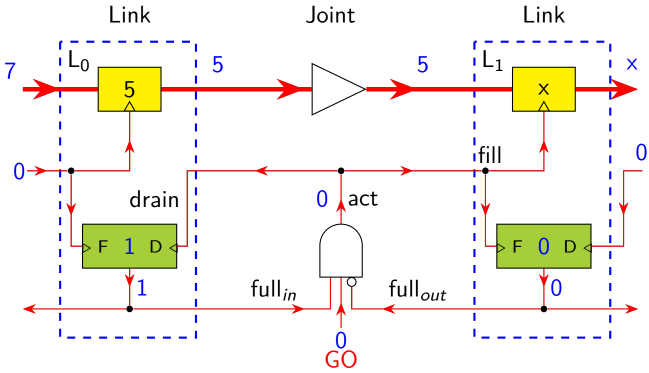


The green boxes represent the instances of the link-control primitive that is added to the DE primitive database.

When a joint acts, three tasks will be executed in parallel:

- transfer data computed from the input links to the output links;
- fill a subset of the output links, leaving them **full**;
- drain a subset of the input links, leaving them **empty**.

The Link-Joint Model

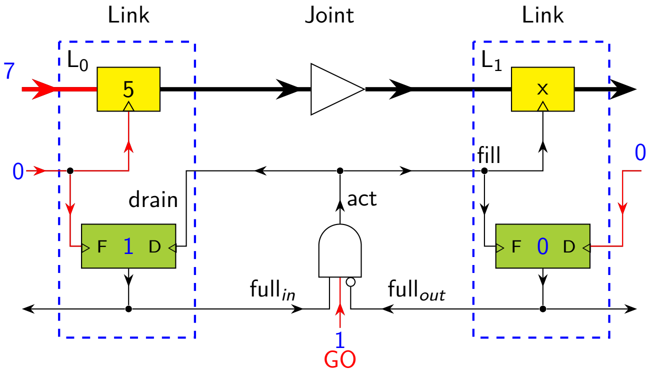


The green boxes represent the instances of the link-control primitive that is added to the DE primitive database.

When a joint acts, three tasks will be executed in parallel:

- transfer data computed from the input links to the output links;
- fill a subset of the output links, leaving them **full**;
- drain a subset of the input links, leaving them **empty**.

The Link-Joint Model

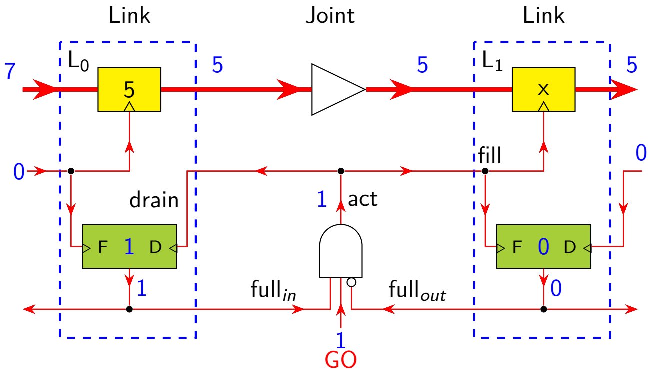


The green boxes represent the instances of the link-control primitive that is added to the DE primitive database.

When a joint acts, three tasks will be executed in parallel:

- transfer data computed from the input links to the output links;
- fill a subset of the output links, leaving them **full**;
- drain a subset of the input links, leaving them **empty**.

The Link-Joint Model

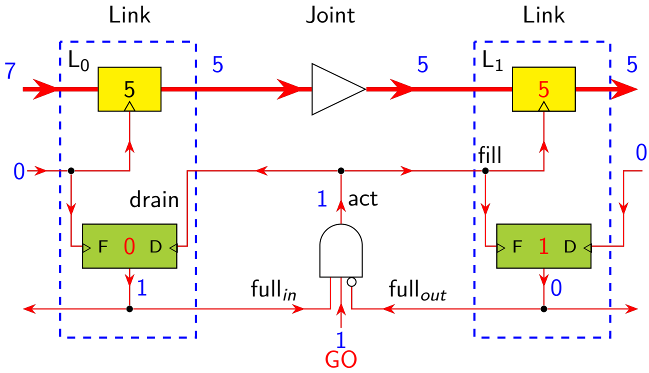


The green boxes represent the instances of the link-control primitive that is added to the DE primitive database.

When a joint acts, three tasks will be executed in parallel:

- transfer data computed from the input links to the output links;
- fill a subset of the output links, leaving them **full**;
- drain a subset of the input links, leaving them **empty**.

The Link-Joint Model

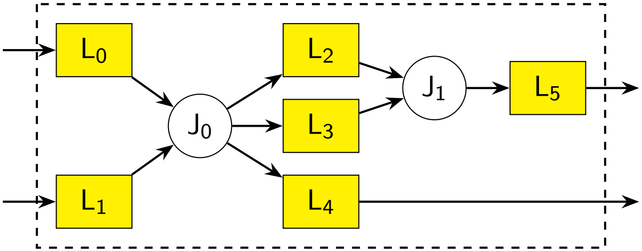


The green boxes represent the instances of the link-control primitive that is added to the DE primitive database.

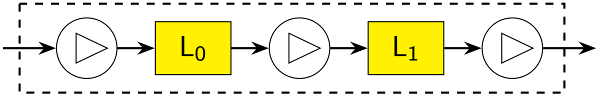
When a joint acts, three tasks will be executed in parallel:

- transfer data computed from the input links to the output links;
- fill a subset of the output links, leaving them **full**;
- drain a subset of the input links, leaving them **empty**.

Self-Timed Modules



Complex link



Complex joint: a queue of length two, *Q*2

Verification

Objective: Verify the **functional correctness** of self-timed circuit designs.

Approach:

- Formalize the relationship between input and output sequences.
- Develop a hierarchical reasoning approach that avoids exploring internal operations of submodules as well as their interleavings.

Verification

Objective: Verify the **functional correctness** of self-timed circuit designs.

Approach:

- Formalize the relationship between input and output sequences.
- Develop a hierarchical reasoning approach that avoids exploring internal operations of submodules as well as their interleavings.
    - Characterize the **one-step update** on the **future output sequence** of a module from the current inputs and current state of that module. We call this property the single-step-update property.

Verification

Objective: Verify the **functional correctness** of self-timed circuit designs.

Approach:

- Formalize the relationship between input and output sequences.
- Develop a hierarchical reasoning approach that avoids exploring internal operations of submodules as well as their interleavings.
    - Characterize the **one-step update** on the **future output sequence** of a module from the current inputs and current state of that module. We call this property the single-step-update property.
    - The single-step-update property of a module is established hierarchically using the single-step-update properties of its submodules, without exploring the internal structures of the submodules.

Verification

Objective: Verify the **functional correctness** of self-timed circuit designs.

Approach:

- Formalize the relationship between input and output sequences.
- Develop a hierarchical reasoning approach that avoids exploring internal operations of submodules as well as their interleavings.
    - Characterize the **one-step update** on the **future output sequence** of a module from the current inputs and current state of that module. We call this property the single-step-update property.
    - The single-step-update property of a module is established hierarchically using the single-step-update properties of its submodules, without exploring the internal structures of the submodules.
    - The **multi-step input-output relationship** is then proved by **induction** with the single-step-update property.

Outline

1. The DE System
2. Modeling and Verification Approach
3. Case Studies
4. Conclusions

Case Studies

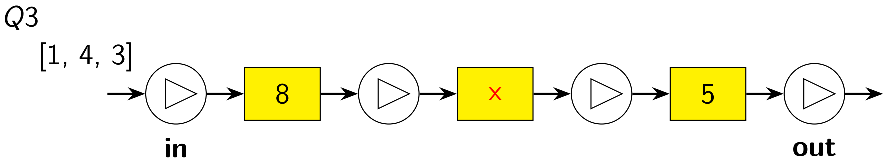
Example 1: A FIFO Circuit

Example 2: A Greatest-Common-Divisor (GCD) Circuit

Example 3: Hierarchical Reasoning

Example 4: Complex Links

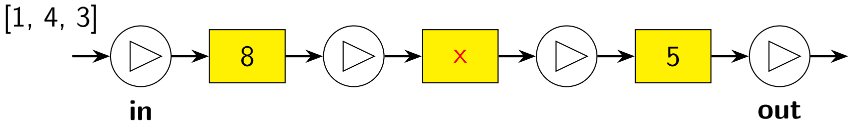
Example 1: A FIFO Circuit



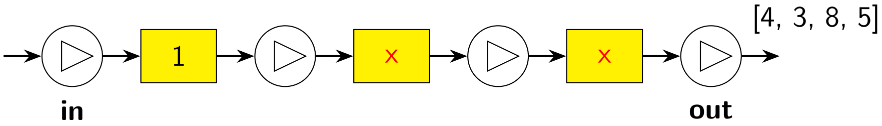
$[1, 4, 3] ++ [8, 5]$

Example 1: A FIFO Circuit

*Q*3



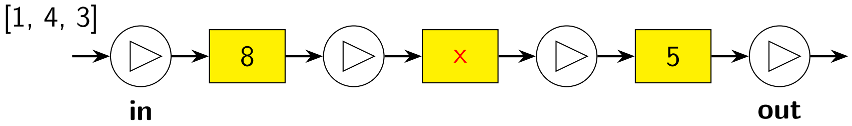
$[1, 4, 3] ++ [8, 5]$



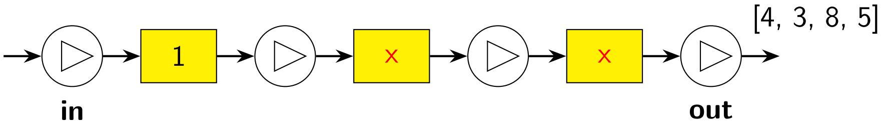
$[1] ++ [4, 3, 8, 5]$

Example 1: A FIFO Circuit

$Q3$



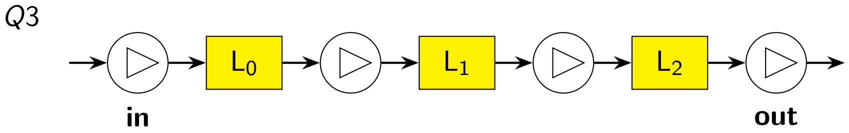
$$[1, 4, 3] ++ [8, 5]$$



$$[1] ++ [4, 3, 8, 5]$$

$$[1] ++ [4, 3, 8, 5] = [1, 4, 3] ++ [8, 5]$$

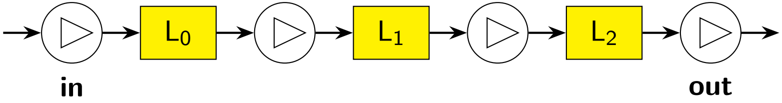
Example 1: A FIFO Circuit



Let in-act and out-act denote the **act** signals from joints **in** and **out**, respectively.

Example 1: A FIFO Circuit

*Q*3

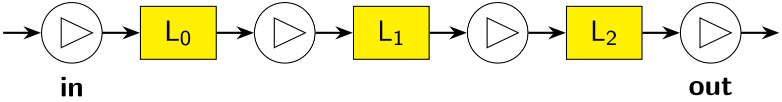


Let in-act and out-act denote the **act** signals from joints **in** and **out**, respectively.

*Q*3 accepts a new data item each time the in-act signal fires. We define in-seq, the **accepted input sequence**, as the sequence of data items that have passed joint **in**.

Example 1: A FIFO Circuit

$Q3$



Let in-act and out-act denote the **act** signals from joints **in** and **out**, respectively.

$Q3$ accepts a new data item each time the in-act signal fires. We define in-seq, the **accepted input sequence**, as the sequence of data items that have passed joint **in**.

Similarly, we define out-seq, the **valid output sequence**, as the sequence of data items that have passed through joint **out** while out-act fires.

Example 1: A FIFO Circuit

The relationship between $Q3$'s in-seq and out-seq.

$$q3\$extract(q3\$run(input\text{-}list, st, n)) ++ out\text{-}seq = in\text{-}seq ++ q3\$extract(st)$$

```
$q3\$run(input\text{-}list, st, n) :=$
  if $(n \leq 0)$ $st$
  else
    $q3\$run(tail(input\text{-}list),$
            $q3\$step(head(input\text{-}list), st),$ // Return the next state of Q3
            $n - 1)$
```

The extraction function ***q3$extract(st)*** extracts valid data from state ***st*** of $Q3$, i.e., extracts data from links that are **full** at state ***st***.

Example 1: A FIFO Circuit

The relationship between $Q3$'s in-seq and out-seq.

$$q3\$extract(q3\$run(input\text{-}list, st, n)) ++ out\text{-}seq = in\text{-}seq ++ q3\$extract(st)$$

```
$q3\$run(input\text{-}list, st, n) :=$
  **if** $(n \leq 0)$ $st$
  **else**
    $q3\$run(tail(input\text{-}list),$
          $q3\$step(head(input\text{-}list), st),$ // Return the next state of Q3
          $n - 1)$
```

The extraction function ***q3$extract(st)*** extracts valid data from state ***st*** of $Q3$, i.e., extracts data from links that are **full** at state ***st***.

out-seq = *in-seq* when the initial and final states contain no valid data.

Example 1: A FIFO Circuit

$$\begin{aligned} q3\$extract(q3\$run(input\text{-}list, st, n)) \mathbin{++} out\text{-}seq = \\ in\text{-}seq \mathbin{++} q3\$extract(st) \end{aligned} \tag{1}$$

Our ACL2 proof of (1) uses **induction** and the following single-step-update property of $Q3$ as a supporting lemma,

$$q3\$extract(q3\$step(input, st)) = q3\$extracted\text{-}step(input, st) \tag{2}$$

Example 1: A FIFO Circuit

$$q3\$extract(q3\$run(input\text{-}list, st, n)) ++ out\text{-}seq = in\text{-}seq ++ q3\$extract(st) \quad (1)$$

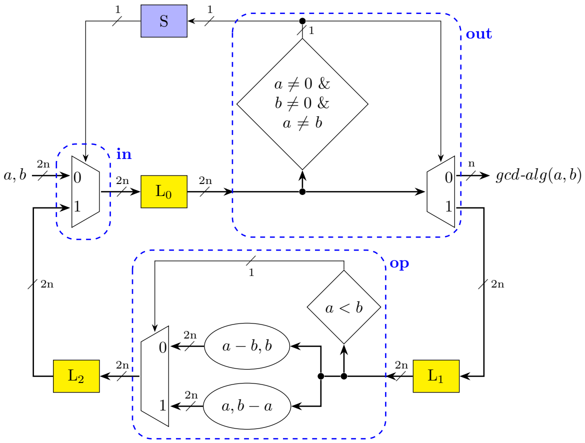
Our ACL2 proof of (1) uses **induction** and the following single-step-update property of $Q3$ as a supporting lemma,

$$q3\$extract(q3\$step(input, st)) = q3\$extracted\text{-}step(input, st) \quad (2)$$

where $q3\$extracted\text{-}step(input, st) :=$

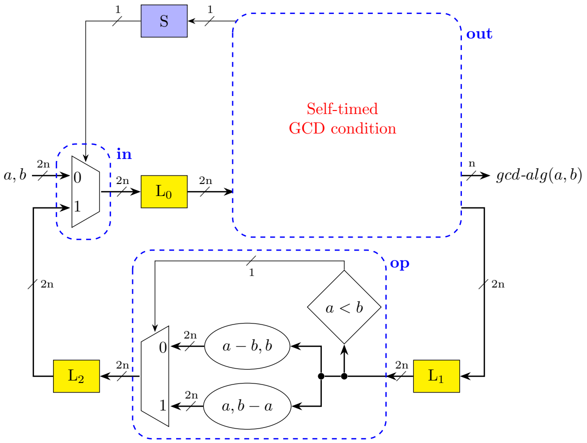
$$\begin{cases} q3\$extract(st), \textbf{if } in\text{-}act = nil \wedge out\text{-}act = nil \\ [input.data] ++ q3\$extract(st), \textbf{if } in\text{-}act = t \wedge out\text{-}act = nil \\ remove\text{-}last(q3\$extract(st)), \textbf{if } in\text{-}act = nil \wedge out\text{-}act = t \\ [input.data] ++ remove\text{-}last(q3\$extract(st)), \textbf{otherwise} \end{cases}$$

Example 2: A Greatest-Common-Divisor (GCD) Circuit



```
gcd-alg(a, b) :=
while (a ≠ 0) ∧ (b ≠ 0) ∧ (a ≠ b) do
    if (a < b)
    then b := b − a
    else a := a − b
return
    if (a = 0) then b else a
```

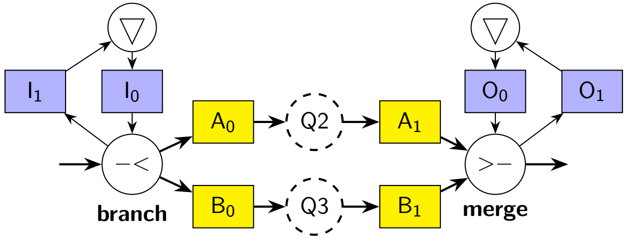
Example 3: Hierarchical Reasoning



```
gcd-alg(a, b) :=
while (a ≠ 0) ∧ (b ≠ 0) ∧ (a ≠ b) do
    if (a < b)
    then b := b − a
    else a := a − b
return
    if (a = 0) then b else a
```

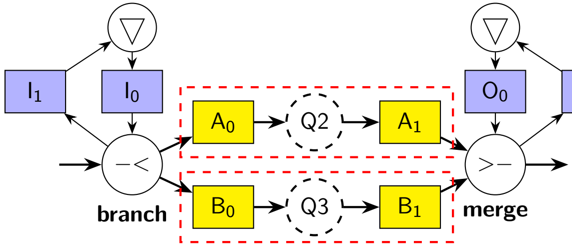
Example 4: Complex Links

RR



Example 4: Complex Links

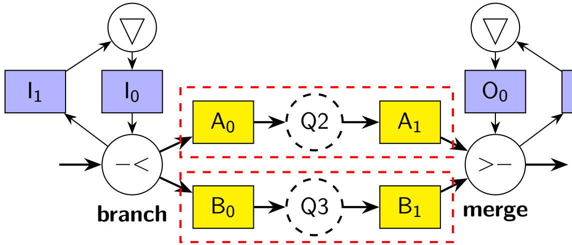
RR



Abstracting two queues ($A_0 \rightarrow Q2 \rightarrow A_1$) and ($B_0 \rightarrow Q3 \rightarrow B_1$) as two complex links makes reasoning more efficient by reducing case splits in proving the invariant as well as the single-step-update property for *RR*.

Example 4: Complex Links

RR



Abstracting two queues ($A_0 \rightarrow Q2 \rightarrow A_1$) and ($B_0 \rightarrow Q3 \rightarrow B_1$) as two complex links makes reasoning more efficient by reducing case splits in proving the invariant as well as the single-step-update property for *RR*.

The verification time of *RR* is reduced from more than 23.5 minutes to 14 seconds by using the complex links.

Outline

1 The DE System

2 Modeling and Verification Approach

3 Case Studies

4 Conclusions

Conclusions

We have presented a framework for formally modeling and verifying self-timed circuit designs using the DE system.

We have developed a hierarchical reasoning method that is capable of verifying the **functional correctness** of self-timed circuit designs at large scale.

This work has also provided a library for analyzing self-timed systems in ACL2.

We model self-timed systems as networks of links communicating with each other locally via joints, using the link-joint model.

We model the **non-determinism of event-ordering** in self-timed circuits by associating each joint with an external go signal that, when disabled, prevents a joint from **firing**.

Timeline

- **Spring 2018 – Fall 2018:** Enhance automation of our framework.
- **Spring 2018 – Summer 2018:** Verify self-timed circuit models that include FCFS arbitrated merge operations.
- **Fall 2018:** Verify a self-timed serial adder model using our new approach.
- **Fall 2018:** Demonstrate compositionality by proving that the functionality of *gcd* is preserved when replacing its combinational ripple-carry-adder sub-circuit with a functionally-equivalent, self-timed serial adder.
- **Spring 2019:** Dissertation writing and final defense.

Publications

Cuong Chau, Warren A. Hunt Jr., Matt Kaufmann, Marly Roncken, and Ivan Sutherland
Data-Loop-Free Self-Timed Circuit Verification
In the 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2018. To appear.

Marly Roncken, Ivan Sutherland, Chris Chen, Yong Hei, Warren Hunt Jr., and **Cuong Chau**, with Swetha Mettala Gilla, Hoon Park, Xiaoyu Song, Anping He, and Hong Chen
How to Think about Self-Timed Systems
In the IEEE Asilomar Conference on Signals, Systems, and Computers, 2017. To appear.

Cuong Chau, Warren A. Hunt Jr., Marly Roncken, and Ivan Sutherland
A Framework for Asynchronous Circuit Modeling and Verification in ACL2
In the 13th Haifa Verification Conference (HVC), 2017, pp. 3-18.

References

C. Chau, W. A. Hunt Jr., M. Roncken, and I. Sutherland (2017)
A Framework for Asynchronous Circuit Modeling and Verification in ACL2
HVC 2017, 3 – 18.

W. A. Hunt Jr. (2000)
The DE Language
Computer-Aided Reasoning: ACL2 Case Studies, Kluwer Academic Publishers Norwell, MA, USA, 151 – 166.

M. Roncken, S. Gilla, H. Park, N. Jamadagni, C. Cowan, I. Sutherland (2015)
Naturalized Communication and Testing
ASYNC 2015, 77 – 84.

M. Roncken, I. Sutherland, C. Chen, Y. Hei, W. Hunt Jr., C. Chau, S. M. Gilla, H. Park, X. Song, A. He, and H. Chen (2017)
How to Think about Self-Timed Systems
Asilomar 2017, to appear.

A. Slobodova, J. Davis, S. Swords, and W. Hunt Jr. (2011)
A Flexible Formal Verification Framework for Industrial Scale Validation
MEMOCODE 2011, 89 – 97.

Questions?