FORMALLY VERIFIED
HIGH-LEVEL SYNTHESIS OF
ELASTIC CIRCUITS

Tony Law
tony.law@inria.fr
tonylaw.fr

Supervised by Sandrine Blazy and Delphine Demange

Université de Rennes - IRISA

2023/11/16 – Journées du GDR GPL – LVP MTV2
1. Context & Goals

2. Current work

3. Conclusion & Future work
1. Context & Goals
FIELD PROGRAMMABLE GATE ARRAY (FPGA)

- Reconfigurable hardware
- Design specialized hardware without the cost of ASICs⁰

⁰Application-specific integrated circuit
**HARDWARE DESIGN PROCESS AND DIFFICULTIES**

- **Closed-source** tools, users are expected to **trust** them
- Widely used HDLs (Verilog, VHDL) are very low-level
  - \( \approx \) assembly language for circuits
High-level synthesis (HLS)

- Compilation of a high-level specification (e.g. C code) into HDL code
- New process on top of “classical” design process
- Goal: speed up development
  - use higher-level code
  - leverage software development tools
GOAL

Design an HLS compiler
Design a verified HLS compiler
GOAL

Design a verified HLS compiler

(Verify that compiled circuits behave like the original program)
Design a **verified** HLS compiler

(Verify that compiled circuits behave like the original program)
Design a verified HLS compiler

which **exploits** hardware parallelism
EXPLOITING HARDWARE PARALLELISM

Both ends use different execution models.

We need to adapt one into another.
CONTROL-DRIVEN EXECUTION MODEL
(Campbell et al., 1993)

Focus on **sequences** of instructions.
Control-flow dictates how to update state.
Unless explicit, not parallel.
Focus on sequences of instructions. Control-flow dictates how to update state. Unless explicit, not parallel.
Focus on **sequences** of instructions.
Control-flow dictates how to update state.
Unless explicit, **not parallel**.
**CONTROL-DRIVEN EXECUTION MODEL**
(Campbell et al., 1993)

Focus on **sequences** of instructions. Control-flow dictates how to update state. Unless explicit, **not parallel**.
Focus on sequences of instructions. Control-flow dictates how to update state. Unless explicit, not parallel.
CONTROL-DRIVEN EXECUTION MODEL
(Campbell et al., 1993)

Focus on sequences of instructions. Control-flow dictates how to update state. Unless explicit, not parallel.
Focus on **sequences** of instructions.
Control-flow dictates how to update state.
Unless explicit, **not parallel**.
Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
**Data-driven execution model**
(Campbell et al., 1993)

Focus on connections between operations. Data tokens flow from one operation to another. Operations are parallelizable.
Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
Data-driven execution model
(Campbell et al., 1993)

Focus on connections between operations. Data tokens flow from one operation to another. Operations are parallelizable.
Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
DATA-DRIVEN EXECUTION MODEL

(Campbell et al., 1993)

Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
**DATA-DRIVEN EXECUTION MODEL**

(Campbell et al., 1993)

Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
Focus on connections between operations. Data tokens flow from one operation to another. Operations are parallelizable.
DATA-DRIVEN EXECUTION MODEL
(Campbell et al., 1993)

Focus on connections between operations. Data tokens flow from one operation to another. Operations are parallelizable.
Focus on connections between operations. Data tokens flow from one operation to another. Operations are parallelizable.
Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
**DATA-DRIVEN EXECUTION MODEL**
(Campbell et al., 1993)

Focus on *connections* between operations. Data tokens flow from one operation to another. Operations are *parallelizable*. 
**Data-driven execution model**
(Campbell et al., 1993)

Focus on **connections** between operations. Data tokens flow from one operation to another. Operations are **parallelizable**.
DIRECTLY COMPILe TO VHDL?

VHDL is very low-level
DIRECTLY COMPILe TO VHDL?

VHDL is very low-level

Target a higher-level HDL instead
DYNAMATIC
(Josipović et al., 2018)

- HLS toolkit from EPFL
- Designed around Elastic Circuits (ECs)\(^1\)

\(^1\)In recent litterature, they are called dataflow circuits
ELASTIC COMPONENTS

- Library of components implemented in VHDL
- Semantics defined as token consumption / production rules
  - Firing rules (Dennis, 1974)
for i in 0..100
    d = A[i] - B[i]
    if (d ≥ 0)
        s += d
for $i$ in $0..100$

$d = A[i] - B[i]$

if ($d \geq 0$)

$s += d$
for $i$ in 0..100
    $d = A[i] - B[i]$
    if ($d \geq 0$)
        $s += d$
for i in 0..100
    d = A[i] - B[i]
    if (d \geq 0)
        s += d
EC PROPERTIES

• Latency insensitivity
  • No need of precise timings
  • Dynamic scheduling
• Data-driven
  • Tokens flow in circuit
  • Components work independently
• Trade-off: higher area/resource usage
FROM CFG TO EC

We need an IR
FROM CFG TO EC

We need an IR

- Need an IR to transition from control to data-driven
FROM CFG TO EC
We need an IR

• Need an IR to transition from control to data-driven
• Rest of the talk: focus on its formalization
2. CURRENT WORK
CURRENT WORK
Formalization of the IR

• Circuit definition and representation
• Operational semantics
• Determinism: a fundamental property of ECs
WHAT IS A CIRCUIT?

• Network of stateless components
• Component communicate via tokens
  • Communication channels = unbounded FIFOs
• State of a circuit = state of its FIFOs
**What is a circuit?**

- Network of stateless components
- Component communicate via tokens
  - Communication channels = unbounded FIFOs
- State of a circuit = state of its FIFOs
**What is a circuit?**

- Network of stateless components
- Component communicate via tokens
  - Communication channels = unbounded FIFOs
- State of a circuit = state of its FIFOs
**WHAT IS A CIRCUIT?**

- Network of stateless components
- Component communicate via tokens
  - Communication channels = unbounded FIFOs
- State of a circuit = state of its FIFOs
CIRCUIT REPRESENTATION

Representation = syntax of the connection description

Two representations: graph-based and language-based
CIRCUIT GRAPHS (CG)

- Natural representation
- Directed graph structure
  - Nodes = stateless components
  - Edges = unbounded FIFOs
- State = state of all FIFOs
**Circuit Graphs (CG)**

- Natural representation
- Directed graph structure
  - Nodes = stateless components
  - Edges = unbounded FIFOs
- State = state of all FIFOs

Problem: difficult to reason about!

No "natural" principle of induction

Tony Law
Current work - Definitions and syntax
CIRCUIT GRAPHS (CG)

- Natural representation
- Directed graph structure
  - Nodes = stateless components
  - Edges = unbounded FIFOs
- State = state of all FIFOs
CIRCUIT GRAPHS (CG)

- Natural representation
- Directed graph structure
  - Nodes = stateless components
  - Edges = unbounded FIFOs
- State = state of all FIFOs

Problem: difficult to reason about!
- No “natural” principle of induction
CIRCUIT LANGUAGE (CL)

- Inductive definition
- State
  - Inputs & Outputs
  - FIFOs of feedbacks
- Induction principle available
CIRCUIT LANGUAGE (CL)

- Inductive definition
- State
  - Inputs & Outputs
  - FIFOs of feedbacks
- Induction principle available
**THEOREM: REPRESENTATION EQUIVALENCE**

*CL* is able to represent all of *CG*
Operational semantics

Semantics of circuits

= 

Semantics of stateless components

(representation-independent)

+ 

State update

(representation-dependent)
SEMANTICS OF STATELESS COMPONENTS

Defined by firing rules: Inputs → Outputs
SEMANITICS OF CG

- Exactly one rule: component firing rule + state update
  - No information on circuit structure
Semantics of CL

Example rules

Every step, exactly one component fires
THEOREM: BISIMULATION

CG and CL are semantically equivalent
Determinism: Why?

- Required for correctness of ECs
  - Only one execution possible from a given starting state
  - No token reordering
- Our IR: Not deterministic
  - No notion of time steps
  - At best, confluent\(^2\)

\(^2\) Only if all components are deterministic
**THEOREM: CONFLUENCE**

Two executions starting from the same state can reach the same state.
**Theorem: Confluence**

Two executions starting from the same state can reach the same state.
**Theorem: Confluence**

Two executions starting from the same state can reach the same state.
THEOREM: CONFLUENCE

Two executions starting from the same state can reach the same state
THEOREM: CONFLUENCE

Two executions starting from the same state can reach the same state.
**Theorem: Confluence**

Two executions starting from the same state can reach the same state.

Image description:

- The diagram illustrates a cycle of processes where two paths converge to a common state, indicating confluence.
- The processes are represented by labeled nodes connected by directed edges, showing the flow of computation.
- The states at each node are depicted with various symbols, indicating different states or outcomes.

This visual representation supports the textual explanation by showing how different execution paths from the same starting point lead to the same end state, emphasizing the concept of confluence.
THEOREM: CONFLUENCE

Two executions starting from the same state can reach the same state i.e.

The execution order of components does not matter
AN INTERMEDIATE LEMMA

Consuming from the “same” input stream will produce the “same” outputs
AN INTERMEDIATE LEMMA

Consuming from the “same” input stream will produce the “same” outputs
AN INTERMEDIATE LEMMA

Consuming from the “same” input stream will produce the “same” outputs
AN INTERMEDIATE LEMMA

Consuming from the “same” input stream will produce the “same” outputs
AN INTERMEDIATE LEMMA

Consuming from the “same” input stream will produce the “same” outputs
3. **Conlusion & Future Work**
CONCLUSION
Current state of the project

- Formalization of an IR for circuits
  - Two representations
  - Operational semantics for both

- Currently proving confluence
  - Irrelevance of execution order
FUTURE WORK

- Validate my IR w.r.t. Dynamatic
  - Build an interpreter
  - Check that it produces the same results
FUTURE WORK

- Validate my IR w.r.t. Dynamatic
  - Build an interpreter
  - Check that it produces the same results
- Control-driven to data-driven
  - Start from a GSA form instead of a CFG
FUTURE WORK

- Validate my IR w.r.t. Dynamatic
  - Build an interpreter
  - Check that it produces the same results
- Control-driven to data-driven
  - Start from a GSA form instead of a CFG
- Formalize hardware ECs
  - Embed time within formalization
After HTL translation, (Herklotz et al., 2021)

- Build an FSM on top the CFG
- One instruction $\approx$ one state
Elastic Protocol
(Josipović et al., 2018), adapted from (Cortadella et al., 2005)

- Two control signals (valid and ready), one data signal.
- Data transfer only occurs when valid and ready are both asserted on the same clock cycle.
- Can be modelled as a token transfer
STATIC SCHEDULING

Instructions are executed according to a schedule fixed at compile time.
Dynamic Scheduling

- No predetermined schedule
- Dynamism achieved by communicating
Optimizations in Dynamatic

Multiple ways to optimize:

- mix static & dynamic circuits
- correctly size LSQ
- insert or remove buffers to increase token throughput
  - can be done after building the EC
  - could be a posteriori validated

(Josipović et al., 2018)
**ON MEMORY**

- In Dynamatic, memory accesses are handled by Load Store Queues (LSQs)
  - role: reorder out-of-order memory accesses
  - use a lot of resources
  - require careful timings and sizing to be efficient
- They are also supported in Dynamatic with GSA (Elakhras et al., 2023)
EC CONSTRAINTS ON LOOPS: DEADLOCKS

- Deadlock in EC = stuck tokens
- Combinational loops (i.e. loops without any buffers) will deadlock
- Solved by inserting buffers to break these loops
ON DETERMINISM

- Determinism required to safely insert buffers
- Non-determinism caused by non-deterministic merges
  - Solved in (Josipović et al., 2018) by surrounding with control logic
  - Solved in (Elakhras et al., 2022) by removing them

\[\text{Loop headers excepted}\]

\[\text{a}\]
IMPROVING DATA DEPENDENCY GRAPHS
(Elakhras et al., 2022)

• From SSA-form CFG, extract data-dependencies
  • graph of (producer → consumer) pairs
• Two problems:
  • produce the right data from multiple sources
  • only propagate data that should be consumed
**GATED SSA (GSA)**
(Herklotz et al., 2023)
**INTERCEPTING AND REMOVING EXTRA DATA**

- Insert “switch” nodes
- Remove or pass input to output depending on predicate
THE PROGRAM DEPENDENCE WEB (PDW)
(Ottenstein et al., 1990)

- PDW ≈ Data-dependency graph + GSA + control-flow infos + switches
  - Data-dependency graph → producer / consumer pairs
  - GSA → correctly produce from multiple sources
  - Switch nodes → stop propagating incorrect data
- One syntax, three equivalent execution models
  - control-driven
  - data-driven
  - demand-driven
- Transition from control-driven to data-driven execution model with this representation
REFERENCES


Accepted: 2022-12-06T08:34:54Z.