Petri Nets and Model Checking in Circuit Design

Lana Josipović

December 2023

ETH zürich
Hardware acceleration for high parallelism and energy efficiency
How to perform hardware design?
High-Level Synthesis: From Programs to Circuits

Raise the level of abstraction for hardware design beyond RTL level (VHDL, Verilog)
High-Level Synthesis: From Programs to Circuits

A completely new type of users for HLS!

Software application programmers

A completely new type of applications for HLS!

General-purpose code
A Different Way to Do HLS

Static scheduling (standard HLS tool): decide at compile time when each operation executes

Dynamic scheduling (our HLS approach): decide at runtime when each operation executes

Circuit regulated by a centralized FSM
→ All execution times predetermined and, sometimes, conservative (slow circuit)

Circuit regulated by distributed handshake logic
→ Flexible execution times (fast circuit)
A Different Way to Do HLS

Static scheduling (standard HLS tool): decide at compile time when each operation executes

Dynamic scheduling (our HLS approach): decide at runtime when each operation executes
A Different Way to Do HLS

**Static scheduling (standard HLS tool):** decide at compile time when each operation executes

**Dynamic scheduling (our HLS approach):** decide at runtime when each operation executes
Dynamically Scheduled Circuits

- **Asynchronous circuits**: operators triggered when inputs are available
  - Budiu et al. Dataflow: A complement to superscalar. ISPASS’05.
- **Dataflow, latency-insensitive, elastic**: the *synchronous* version of it
  - Cortadella et al. Synthesis of synchronous elastic architectures. DAC’06.
  - Carloni et al. Theory of latency-insensitive design. TCAD’01.
  - Jacobson et al. Synchronous interlocked pipelines. ASYNC’02.

High-level synthesis of dynamically scheduled circuits
HLS of Dynamically Scheduled Circuits
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

Resource sharing

Mul 1
Mul 2
Mul 1/2
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

Resource sharing

Out-of-order memory

Reaping the benefits of dynamic scheduling

Speculative execution
Dataflow Circuits

• We use the **SELF (Synchronous ELastic Flow)** protocol
  – Cortadella et al. Synthesis of synchronous elastic architectures. DAC’06.
• Every component communicates via a pair of handshake signals
• **Make scheduling decisions at runtime**
  – As soon as all conditions for execution are satisfied, an operation starts
Dataflow Circuits

• We use the **SELF (Synchronous ELastic Flow)** protocol
  – Cortadella et al. Synthesis of synchronous elastic architectures. DAC’06.

• Every component communicates via a pair of handshake signals

• **Make scheduling decisions at runtime**
  – As soon as all conditions for execution are satisfied, an operation starts
We use the **SELF (Synchronous ELastic Flow)** protocol
- Cortadella et al. Synthesis of synchronous elastic architectures. DAC’06.

Every component communicates via a pair of handshake signals

**Make scheduling decisions at runtime**
- As soon as all conditions for execution are satisfied, an operation starts

![Dataflow Circuits Diagram](image_url)
Dataflow Components

- Fork
- Join
- Branch
- Merge
Dataflow Components

- Fork
- Join
- Branch
- Merge

Data flow components diagram with arrows indicating flow and connections.
Dataflow Components

- Fork
- Join
- Branch
- Merge
- STORE
Dataflow Components

- Fork
- Join
- Merge
- Branch
Dataflow Components

- Fork
- Join
- Branch
- Merge
Dataflow Components

- Fork
- Join
- Branch
- Merge
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
From Program to Dataflow Circuit

for (i=0; i<N; i++) {
    `hist[x[i]] = hist[x[i]] + weight[i];`
}
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
From Program to Dataflow Circuit

for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 Best Paper Award Nominee
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017
```c
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
```
From Program to Dataflow Circuit

Backpressure due to insufficient token capacity: no pipelining and low performance

Josipović, Ghosal, and Ienne. Dynamically Scheduled High-Level Synthesis. FPGA 2018 Best Paper Award Nominee
Josipović, Brisk, and Ienne. From C to Elastic Circuits. Asilomar 2017
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

Fork
Load
FIFO
Store
ready
stall

Resource sharing

Mul 1
Mul 2
Mul 1/2

Reaping the benefits of dynamic scheduling

Out-of-order memory

Speculative execution

Load
LD
ST
Memory

Save
Fork
Speculator
Commit

Save
Fork
Commit
Inserting Buffers

```c
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
```

Buffers as registers to break combinational paths

---

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 Best paper award
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022
Inserting Buffers

for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}

Buffers as FIFOs to regulate throughput

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 Best paper award
Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022
Inserting Buffers

```c
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
```

BEFORE (without buffers)

Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 Best paper award

Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022
Josipović, Sheikhha, Guerrieri, Ienne, and Cortadella. Buffer Placement and Sizing for High-Performance Dataflow Circuits. FPGA 2020 Best paper award

Rizzi, Guerrieri, Ienne, and Josipović. A Comprehensive Timing Model for Accurate Frequency Tuning in Dataflow Circuits. FPL 2022
Inserting Buffers

Now (with buffers)

Represent program loops as choice-free Petri nets
- Analyze average token flow through the circuit (continuous Petri net)
- Determine buffer positions & sizes (token capacity)
- Maximize throughput for a target clock period

for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
HLS of Dynamically Scheduled Circuits

**Catching up with static HLS**

- Pipelining

```
Fork
Load
FIFO
Store
```

- Resource sharing

```
Mul 1
Mul 2
Mul 1/2
```

**Reaping the benefits of dynamic scheduling**

- Out-of-order memory

```
Load
Store
```

- Speculative execution
Saving Resources through Sharing

- Static HLS: share units between operations which execute in different clock cycles
- Dynamic HLS: share units based on their average utilization with tokens

```java
for (i = 0; i < N; i++) {
    a[i] = a[i]*x;
    b[i] = b[i]*y;
}
```

Saving Resources through Sharing

- Static HLS: share units between operations which execute in **different clock cycles**
- Dynamic HLS: share units based on their **average utilization** with tokens

```plaintext
for (i = 0; i < N; i++) {
    a[i] = a[i]*x;
    b[i] = b[i]*y;
}
```

**Units fully utilized**
(high throughput, II = 1)

Sharing not possible without damaging throughput

**Use choice-free Petri net model to decide what to share**

Josipović, Marmet, Guerrieri, and Ienne. Resource Sharing in Dataflow Circuits. FCCM 2022. **Best Paper Award Nominee**
Saving Resources through Sharing

- Static HLS: share units between operations which execute in **different clock cycles**
- Dynamic HLS: share units based on their **average utilization** with tokens

```plaintext
for (i = 0; i < N; i++) {
    a[i] = a[i]*x;
    b[i] = b[i]*y;
}
```

Sharing possible without damaging throughput

Units underutilized (low throughput, II = 2)

Use choice-free Petri net model to decide what to share

HLS of Dynamically Scheduled Circuits

Catching up with static HLS

- Pipelining
  - Fork
  - Load
  - RF
  - FIFO
  - ready
  - stall
  - Store

- Resource sharing
  - Mul 1
  - Mul 2
  - Mul 1/2

Reaping the benefits of dynamic scheduling

- Out-of-order memory
  - LSQ
  - Load
  - Store
  - Memory

- Speculative execution
  - Save
  - Speculator
  - Fork
  - Commit
We Need a Load-Store Queue (LSQ)!

- Processor LSQs keep dependent memory accesses in the original program order

```
loop: lw $t2, 0($t4)
lw $t3, 100($t4)
mul $t5, $t2, $t3
addi $t5, $t5, $t1
sw $t5, 100($t4)
addi $t1, $t1, 4
bne $t6, $t1, loop
```

- Application-specific LSQs for dataflow circuits

```
for (i=0; i<N; i++) {
    hist[x[i]] = hist[x[i]] + weight[i];
}
```
HLS of Dynamically Scheduled Circuits

Catching up with static HLS

Pipelining

Resource sharing

Mul 1
Mul 2
Mul 1/2

Reaping the benefits of dynamic scheduling

Out-of-order memory

Speculative execution
Speculation in Dataflow Circuits

- Contain speculation in a region of the circuit delimited by special components
  - Issue speculative tokens (pieces of data which might or might not be correct)
  - Squash and replay in case of misspeculation
Dynamatic: An Open-Source HLS Compiler

- From C/C++ to synthesizable dataflow circuit description

Reduced execution time in irregular benchmarks (speedup of up to 14.9X)

But... dataflow computation is resource-expensive!
The Cost of Dataflow Computation

```c
for (i=0; i<N; i++) {
    a[i] = a[i]*c;
}
```
The Cost of Dataflow Computation

Distributed dataflow handshake mechanism: resource and frequency overhead
The Cost of Dataflow Computation

Do we need expensive dataflow logic *everywhere*?
Removing Excessive Dynamism

Data is never stalled

Possible stall
Removing Excessive Dynamism

Data is never stalled

Restrict the generality of dataflow logic whenever it is not needed

Possible stall
Removing Excessive Dynamism

How to guarantee correctness of simplifications for any possible circuit behavior?

Data is never stalled
How to Guarantee Correctness?

Our goal: a formal verification framework for reducing the hardware complexity of dataflow circuits

**Functional verification**
*Covers representative behaviors*

**Functional verification is inefficient and non-exhaustive**

**Formal verification**
*Covers all behaviors*

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.
Proving Properties to Eliminate Excessive Dynamism

For each channel: prove the absence of backpressure (remove logic to compute the ready signal)
AG (valid \rightarrow ready)

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.
For each channel: prove the **absence of backpressure** (remove logic to compute the ready signal)

\[ \text{AG (valid} \rightarrow \text{ready)} \]

For each pair of channels: prove **trigger equivalence** (remove logic to compute one of the valid signals)

\[ \text{AG (valid}_1 \leftrightarrow \text{valid}_2) \]
For each channel: prove the **absence of backpressure**
(remove logic to compute the ready signal)
AG (valid $\rightarrow$ ready)

For each pair of channels: prove **trigger equivalence**
(remove logic to compute one of the valid signals)
AG (valid1 $\leftrightarrow$ valid2)

**Up to 50% area reduction without a performance penalty**

But it is very slow (~hrs)...

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.
Ensuring Scalability by Compositional Verification

- **Decompose circuit** into regions whose properties can be verified independently
- **Abstract the complexity** of other regions into simpler nodes that have the same properties as the circuit they encapsulate

```plaintext
for (i = 0; i < N; i++)
    ...

for (i = 0; i < N; i++)
    ...
```

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.
Ensuring Scalability by Compositional Verification

- **Decompose circuit** into regions whose properties can be verified independently
- **Abstract the complexity** of other regions into simpler nodes that have the same properties as the circuit they encapsulate

```
for (i = 0; i < N; i++)
   ...
for (i = 0; i < N; i++)
   ...
```

Up to 8X reduction in checking time

Xu, Murphy, Cortadella, and Josipović. Eliminating excessive dynamism of dataflow circuits using model checking. FPGA 2023.
Enable diverse users to accelerate compute-intensive applications on hardware platforms.
MSc & BSc Projects and Theses

• Use **Petri nets** to describe circuits and their behaviors
  – Component modelling
  – Performance and area optimizations

• Use **model checking** to prove circuit properties and improve their quality
  – Checking more complex properties
  – Dealing with scalability issues

• And many other topics...

• Check link on last slide for (non-exhaustive) list of projects!

---

Come work with us! 😊
MSc Course in Spring 2024: Synthesis of Digital Circuits

• Algorithms, tools, and methods to generate circuits from high-level programs
  – How does ‘classic’ HLS work?
• Recent advancements and current challenges of HLS for FPGAs
  – What is HLS still missing?
• Course organization
  – First part: lectures+exercises
  – Second part: practical work + seminar-like discussions
• Link to Course Catalogue info (2024)

Hope to see you there! ☺
DYNAMO: Digital Systems and Design Automation Group

dynamo.ethz.ch
ljosipovic@ethz.ch

Project list 2024