Compositional Dataflow Circuits

Stephen A. Edwards  Richard Townsend  Martha A. Kim

Columbia University

MEMOCODE, Vienna, Austria, October 1, 2017
\[
gcd(a, b) = \begin{cases} 
  a & \text{if } a = b \\
  \gcd(a, b - a) & \text{else if } a < b \\
  \gcd(a - b, b) & \text{else}
\end{cases}
\]
\[ \gcd(a, b) = \]

  if \( a = b \)
      \( a \)
  else if \( a < b \)
      \( \gcd(a, b - a) \)
  else
      \( \gcd(a - b, b) \)
\[
gcd(a, b) = \begin{cases} 
  a & \text{if } a = b \\
  \gcd(a, b - a) & \text{else if } a < b \\
  \gcd(a - b, b) & \text{else}
\end{cases}
\]
\[ \text{gcd}(a, b) = \]
if \( a = b \)
    \( a \)
else if \( a < b \)
    \( \text{gcd}(a, b - a) \)
else
    \( \text{gcd}(a - b, b) \)
gcd\((a, b) =
\quad \text{if } a = b \quad a
\quad \text{else if } a < b \quad \text{gcd}(a, b - a)
\quad \text{else} \quad \text{gcd}(a - b, b)
}
\[ \text{gcd}(a, b) = \]
\[ \text{if } a = b \]
\[ a \]
\[ \text{else if } a < b \]
\[ \text{gcd}(a, b - a) \]
\[ \text{else} \]
\[ \text{gcd}(a - b, b) \]
\( \text{gcd}(a, b) = \)
if \( a = b \)
\( a \)
else if \( a < b \)
\( \text{gcd}(a, b - a) \)
else
\( \text{gcd}(a - b, b) \)
\[ \text{gcd}(a, b) = \]
\[ \text{if } a = b \]
\[ a \]
\[ \text{else if } a < b \]
\[ \text{gcd}(a, b - a) \]
\[ \text{else} \]
\[ \text{gcd}(a - b, b) \]
Patience Through Handshaking

Want *patient* blocks to handle delays from

- Memory systems
- Data-dependent computations

---

- Full buffers
- Shared resources
- Busy computational units
Patience Through Handshaking

Want *patient* blocks to handle delays from Memory systems Data-dependent computations

<table>
<thead>
<tr>
<th>valid</th>
<th>ready</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Token transferred</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Token valid; held</td>
</tr>
<tr>
<td>0</td>
<td>–</td>
<td>No token to transfer</td>
</tr>
</tbody>
</table>

Latency-insensitive Design (Carloni et al.)
Elastic Circuits (Cortadella et al.)
FIFOs with backpressure
Combinational Function Block

Strict/Unit Rate:
All input tokens required to produce an output

Datapath

Combinational function ignores flow control
Combinational Function Block

Strict/Unit Rate:
All input tokens required to produce an output

Valid network
Output valid if both inputs are valid
Combinational Function Block

Strict/Unit Rate:
All input tokens required to produce an output

Ready network
Input tokens consumed if output token is consumed
(output is valid and ready)
Multiplexer Block

in0 in1 in2

select

out

in0

in1

in2

decoder

select

out
Demultiplexer Block

Diagram of a demultiplexer block with inputs `in`, `select`, and outputs `out0`, `out1`, `out2`.
Buffering a Linear Pipeline (Point 1/4)

Combinational block

Data buffer: Pipeline register with valid, enable

Control Buffer: Register diverts token when downstream suddenly stops

Long Combinational Path (Data + Valid)

Long Combinational Path (Ready)

Cao et al. MEMOCODE 2015

Inspired by Carloni's Latency Insensitive Design (e.g., MEMOCODE 2007)
Buffering a Linear Pipeline (Point 1/4)

Long Combinational Path (Data + Valid)
Buffering a Linear Pipeline (Point 1/4)

Data buffer:
Pipeline register with valid, enable

Long Combinational Path (Data + Valid)
Long Combinational Path (Ready)

Cao et al. MEMOCODE 2015
Inspired by Carloni's Latency Insensitive Design (e.g., MEMOCODE 2007)
Buffering a Linear Pipeline (Point 1/4)

Combinational block
Data buffer:
Pipeline register with valid, enable

Control Buffer:
Register diverts token when downstream suddenly stops

Long Combinational Path (Data + Valid)
Long Combinational Path (Ready)

Inspired by Carloni's Latency Insensitive Design (e.g., MEMOCODE 2007)
Buffering a Linear Pipeline (Point 1/4)

Cao et al. MEMOCODE 2015
Inspired by Carloni’s Latency Insensitive Design (e.g., MEMOCODE 2007)
The Problem with Fork

Combinational Block: inputs ready when both valid & output ready

Oops: Combinational Cycle
This is not compositional
The Problem with Fork

Combinational Block: inputs ready when both valid & output ready

Fork: outputs valid only when all are ready

Oops: Combinational Cycle

This is not compositional
The Problem with Fork

Combinational Block:
inputs ready when both valid & output ready

Fork:
outputs valid only when all are ready

Oops: Combinational Cycle
This is not compositional
The Problem with Fork

Combinational Block: inputs ready when both valid & output ready

Fork: outputs valid only when all are ready

Oops: Combinational Cycle

This is not compositional
The Problem with Fork

Combinational Block:
inputs ready when both valid & output ready

Fork:
outputs valid only when all are ready

Oops: Combinational Cycle
This is not compositional
The Solution to Combinational Loops (Point 2/4)

Allowed: Combinational paths from valid to ready

Prohibited: Combinational paths from ready to valid
The Solution to Combinational Loops (Point 2/4)

Allowed: Combinational paths from valid to ready

Prohibited: Combinational paths from ready to valid
The Solution to Combinational Loops (Point 2/4)

Allowed: Combinational paths from valid to ready

Prohibited: Combinational paths from ready to valid
The Solution to Combinational Loops (Point 2/4)

Allowed: Combinational paths from valid to ready

Prohibited: Combinational paths from ready to valid
The Solution to Fork: A Little State (Point 3/4)

Valid out ignores ready of other outputs

Input consumed once one token sent on every output
The Solution to Fork: A Little State (Point 3/4)

Flip-flop set after token sent suppresses duplicates

Valid output ignores ready of other outputs

Input consumed once one token sent on every output
The Solution to Fork: A Little State (Point 3/4)

Flip-flop set after token sent suppresses duplicates

Valid out ignores ready of other outputs

Input consumed once one token sent on every output
Nondeterministic Merge (Point 4/4)

Share with merge/demux
Two-Way Nondeterministic Merge Block w/ Select

Arbiter

“Two-way fork with multiplexed output selected by an arbiter”
Experiments: Random Buffer Placement

- **GCD(100,2)**
  - Number of buffer pairs: (7 buffers)
  - Completion Time (µs): 0, 750, 1500, 2250

- **21-way Conveyor**
  - Number of buffer pairs: (80 buffers)
  - Completion Time (µs): 0, 1, 2, 3

- **BSN**
  - Number of buffer pairs: (96 buffers)
  - Completion Time (µs): 0, 1, 2, 3

Graphs showing the relationship between the number of buffer pairs and completion time for different buffer placements.
Best Buffering for GCD (Manually Obtained)

Each loop has one of each buffer

- **Data Buffer**
- **Control Buffer**
Summary

Compositional Dataflow Networks as an IR

Patient dataflow blocks with valid/ready handshaking

1. Break downstream, upstream paths w/ two buffer types

2. Avoid comb. cycles: prohibit ready-to-valid paths

3. Add one state bit per output so forks may “race ahead”

4. Tame nondeterministic merge with a select output

Random buffer placement experiments show it works