Adapting Asynchronous Circuits to Operating Conditions by Logic Parameterisation

Andrey Mokhov, Danil Sokolov, Alex Yakovlev
Newcastle University, UK
Designer’s dilemma

• Contradicting requirements:
  o ↑ performance, ↓ energy consumption, ↑ robustness
  o high performance → high energy consumption
  o low energy consumption → low robustness
  o high robustness → low performance

• Competing implementation styles:
  – Synchronous
  – Asynchronous
    • Delay Insensitive (DI)
    • Speed Independent (SI) or Quasi Delay Insensitive (QDI)
    • Bundled data, relative timing assumptions (TA)
    • ...
Energy-efficiency v Robustness

Energy availability

Synchronous

Bundled-data

Delay-insensitive

Uncertainty
Parameterised Circuits

- Static parameters set at test/binning stages
- Dynamic parameters:
  - Power management controller
  - Maintenance mechanisms
Parameterised Circuits: Trivial Approach

- Easy to design!
- Large overheads
- How can we do better?
Parameterised Circuits: Better Approach

• **Goal:**
  – Combine circuit implementations *efficiently*

• **Key observations:**
  – Externally: the circuits have the same interface
  – Internally: the circuits behave *similarly*

• **Solution:** use a model that can capture functional similarities at the circuit level
  – Conditional Partial Order Graphs *almost* fit
Conditional Partial Order Graphs

Diagram showing conditional partial order graphs with nodes labeled as follows:
- Node $x=0$
- Node $y=0$
- Node $x=1$
- Node $y=1$

The diagram includes nodes labeled $a$, $b$, $c$, $d$, $e$ with directed edges indicating conditional relationships and values for $x$, $y$, and their negations ($\overline{x}$, $\overline{y}$).
Conditional Partial Order Graphs

👍 Describe concurrency and causality

👍 Capture similarities in behaviours

👎 Represent families of partial orders
  • acyclic
  • not directly applicable to circuit specification

👉 Is it possible to specify cyclic behaviour using acyclic objects?
  — Yes!
Describing cycles
Describing cycles

diagram of a cycle with labeled nodes and an arrow labeled 'token'
Describing cycles
Describing cycles
Describing cycles

\[ x = 0 \]

\[ x = 1 \]
Describing cycles

Initial state: $x = 1$

Set $x = 0$

Initial state: $x = 1$
Describing cycles

\[ x = 1 \]

d: \overline{x} → a

b: x → c

c → d: \overline{x}

\[ \checkmark \]
Describing cycles

\[ x = 0 \]
Describing cycles

\[ x = 1 \]

\[ d: \bar{x} \]

\[ a \] \quad \checkmark \quad \checkmark

\[ b: x \] \quad \checkmark

\[ c \] \quad \checkmark

\[ \checkmark \]

\[ \checkmark \]
Describing circuits: oscillator

Conditional Signal Graphs (CSGs)
Describing circuits: C-element

\[ a=0 \]
\[ b=0 \]

\[ c=0 \]
Describing circuits: C-element

\[ a=0 \]
\[ b=0 \]
\[ c=0 \]
Describing circuits: C-element

\[ abc = 000 \]
Describing circuits: C-element

\[ abc=000 \]
Describing circuits: C-element

\[ abc = 100 \]
Describing circuits: C-element

\[ abc=110 \]
Describing circuits: C-element
Describing circuits: C-element
Describing circuits: C-element

\[ abc = 001 \]
Describing circuits: C-element

\[ abc=000 \]
Circuit composition

\[
c = ab + c(\alpha + b)
\]
Circuit composition

C-element:
\[ \uparrow \max(a\uparrow, b\uparrow) \]  
\[ \downarrow \max(a\downarrow, b\downarrow) \]

AND-gate:
\[ \uparrow \max(a\uparrow, b\uparrow) \]  
\[ \downarrow \min(a\downarrow, b\downarrow) \]

C/AND-element:
\[ \text{can be chosen in run-time!} \]
Circuit composition

• Algebraic operations with CSGs:
  – Addition (overlay): $G_1 + G_2$
  – Scalar multiplication (encoding): $f \cdot G$

• Composition of C-element and AND-gate:

\[ G = p \cdot G_C + \overline{p} \cdot G_{AND} \]

• General composition of $n$ circuits:

\[ G = f_1 \cdot G_1 + f_2 \cdot G_2 + \cdots + f_n \cdot G_n \]

• Fast structural operation (if encoding is given)
Design flow

- Heterogeneous models and implementation styles
- Not computationally expensive:
  - composition is fast
  - exact encoding is slow but there are fast approximate methods
Example: Read/Write controller

![Diagram of a Read/Write controller](image)

- **Parameters**
  - $r$
  - $a$
  - $r_1$
  - $a_1$
  - $r_2$
  - $a_2$
  - $r_3$
  - $a_3$

- **Clock (if needed)**

- **Data path (bus, registers, etc.)**
Example: possible implementations

Delay Insensitive (DI)
\[ T = d + \max\{d_1, d_2, d_3\} + C_3 \]

Partial Acknowledgement (PA)
\[ T = d + \max\{d_1, d_2\} + C_2 \]

Timing Assumptions (TA)
\[ T = d + I \]

Synchronous (CL)
\[ T = t_{\text{clock}} \]
Example: parameterised controller

Can be switched off by power-gating in TA/CL modes
Example: simulation results

latency, ps

- parameterised controller (optimised for TA mode)
- parameterised controller
- specialised controllers

DI mode
PA mode
TA mode

synchronous mode

robustness, abstract units
Cost of Parameterisation

• Trivial approach:
  – Latency overhead: 120-182%, 122% on average
  – Energy overhead: 151-330%, 201% on average

• Our approach:
  – Latency overhead: 0-40%, 19% on average
  – Energy overhead: 0-98%, 23% on average

• Latency/energy savings at the system level!
Conclusions & Future work

• “Don’t oppose different implementation styles, take advantage of their benefits!” (reviewer X)

• New model for circuit-level description and composition

• Cost of parameterisation is not prohibitive

• Future work:
  – Tool prototyping
  – CSC signals awareness
  – Power-gating mechanisms
Questions?