Problem Definition: understanding process communication

Interacting with each other and the environment,

a top-down fashion.

Linking system synthesis and asynchronous circuit synthesis,

General Problem
Interactions
Common aspect - potential for concurrent communication
- Coordinating concurrent behaviors.
- Resource sharing - bussses, memory, etc.
- Compensating differently timed parts.
- Some characteristics

System Synthesises
Asynchronous Synthesis

• No set method for dealing with interface issues.

• Starting point - specifications expressed at differing levels.

• Only restricted types of choice implementable.

• Many found for hazard-free controller circuits.

• Tools rapidly maturing, but still limited.
Develop a technique for connecting system specifications and toolchain synthesis methods via a design-space exploration tool.

Proposed Method

Design a theory for understanding and manipulating system specifications to help designers to reduce impact of abstraction on performance.

Research Goal
1. Problem Introduction and Proposed Solution

2. Procedure and Design Flow

3. Method Details

4. Contributions, Challenges, Conclusions

5. Timetable

Talk Outline
existing asynchronous method(s).

Finish: Separate and implement remaining behavior using metastability-robust library part. Evaluate.

• Extract arbitration behavior and map to appropriate requirements.

• Analyze and manipulate flow graph to minimize arbitration

(CFG).

• Translate requirements to a Communication Flow Graph

communication behavior.

Start: User provides description specifying high-level

Procedure
parallel actions \| action choice (choose one only) \\
output action \| input action

Notation:

behavioral specification

Desired communication behavior expressed in CSP language

System Description
of Graph.

Source and Sink are vacant nodes, representing start and end.

Edge attributes represent additional sequencing constraints.

Directed by vertex $v$?

Represented by vertex $v$? Proceeds communication action

- $E$, a set of edges. An edge $e$ means communication action

- $\Lambda$, a set of vertices. Each vertex or node represents a

Directed Graph $G = (\Lambda, E)$ with

Communication Flow Graph
design objective(s).

- Rearrange and remap to optimize concurrency and other
  to arbiters.

- Pattern matching on graph substructures to locate and map
  (metastability-robust components).

- Transformation based on underlying implementation style
  sequence preserving transformations.

Graph nodes are grouped into hypergraphs according to
Provided as netlist description.

Complexity, effect on concurrency.

Each part characterized for functionality, area, speed.

q-selects, mutex, arbiters.

Metastability-robust parts including synchronizers, g-hops.

Library Components
System Component Behaviors

5. COP behavior, fair arbitration, access priority
4. arbitration access to shared resource (blocking, nonblocking)
3. safe value sampling
2. synchronization
1. enforcing mutual exclusion

Task: specifying, identifying, and modeling coordination behaviors, including:

Task Specifying, Identifying, and Modeling Coordination Behaviors
Partitioning

Deterministic graph structure remaining after mapping to arbiter components is transformed to description suitable for synthesis with ATACs, or ACK, or other asynchronous tool.
Example: Simple Router
endit
forward packet out port X

decrement addr;
else
send packet out port Y
remove addr from packet;
If addr == 0, then
receive a packet;
Loop forever

/* At each receiving port A and B, do */

Simple Router - FUNCTIONAL DESCRIPTION
except any edges from source node (graph root) assumed.

Edges represent communication dependencies.

In deriving the communication flow graph, initially each named

( [ i, i' ] : B_i ) || ( [ i | i' ] : A_i )

omitting operational details, abstract communication actions.

Simple Router - COMMUNICATION BEHAVIOR
convolutional design. Subsequent groupings can produce same structure as

3×3 Router: showed mapping choices influenced by library parts

for syntheses. Mapped to arbitrary parts and remaining behavior expressed

matching delay, showed how the transformed flow graph gets

Grouping’s effects on several re-optimizing concurrency, or

original spec, showed method for recognizing communication

Simple Router: showed method for deriving flow graph from

Results
Additional Concerns - Flow Graph Annotations

Functional behavior that affects communication flows must be expressed. For example, to distinguish the flow graph representation for the following 2 behaviors.

* (A? ; X! ; B? ; X!) /* wait to do A?, wait to do B? */

* ((A? || B?) ; X!) /* A? and B? nondetermin. order */

Expressing and recognizing nonblocking arbitration.

Annotating communications with priorities.
Expected Contributions

- Asynchronous synthesis
  - Separate and expressing remaining behavior
  - Optimize concurrency
  - Evaluation techniques to guide graph manipulations and parts.

- A method for recognizing and mapping to particular abstract communication flows.

- A graph structure for representing and manipulating required by system specification.

- CSP-variant for expressing communication flows as required.
Asynchronous synthesis methods

Allows optimizing system organization for robust benchmarks.

Method to be used and evaluated on progressively larger system specifications, including existing asynchronous examples and implementations.

Conclusion
IN Degree = 2

OUT Degree = 1

Nodes with

IN Degree = 2

OUT Degree = 1

Buffer

Router

Mutex

Arbiter

OUT Degree = 2

OUT Degree = 1

IN Degree = 2

IN Degree = 1

Graph Topologies for Mapping
Communication Flow Graph

\[ (* (A?; [X! | Y!]) \parallel (* (B?; [X! | Y!]) \]

\[ (A?; [X! | Y!]) \parallel (B?; [X! | Y!]) \]
Group Nodes by Edges Out

Maps to Arbiter

Aberr
3x3 Router
Pout : tristable mutex
Y : tristable mutex
X : 2 input mutex

3x3 Router - Another Grouping

**Group by Out Edges**

**3x3 Router - Another Grouping**
3x3 Router - Another Grouping - cont'd

Group by In Edges

\[ (*) \text{APin?} ; (X! | Y! | Pout!) \]

\[ (*) \text{B2?} ; (Y! | Pout!) \]