Outline

- Control Circuits
- Petri Nets & Signal Transition Graphs
  - Properties
  - Common PN/STG fragments
- Synthesis of SI control circuits
  - State Encoding
  - Next-state functions
  - Implementation
- Synthesis tool: Petrify
Control Circuits

- Control logic essential part of asynchronous circuits
  - How to specify?
  - How to implement?
Petri Nets (PNs)

- For modelling concurrent systems
- Directed graph with nodes and arcs
- Nodes: places, transitions
- Places can be marked with tokens
- Transition is enabled (allowed to fire) if all input places have tokens
- When a transition fires:
  - Token removed from all input places
  - Token added to each output place
STGs

- Restricted subclass of petri nets
- PN transitions = signal transitions
- Simple places omitted (places with a single input and a single output)
- Places/arcs represent causal relationships between signal transitions
- Marking represents circuit state
PN/STG - Example

Muller C-gate

Source: [Sparso 06]
Properties of STGs I

- **Input free choice**
  - Alternative transitions only controlled by mutually exclusive inputs

- **1-bounded**
  - Max. one token per place

- **Liveness**
  - STG is live iff from every reachable marking, every transition can eventually fire
Typical PN/STG Fragments

- Choice
- Merge
- Fork
- Join
PN/STG Fragments - Example

Source: [Sparso 06]
Properties for STGs II

- STGs can be implemented as speed-independent circuits. Requirements:
  - Consistent state assignment
    - In any execution, any transition alternates between rising and falling
  - Persistency
    - Enabled signals will eventually fire, cannot be disabled by other transition
  - Complete state coding (CSC)
    - Different markings must represent different states
## Speed-Independence (SI)

- **Delay-model: speed-independence**
  - Arbitrary gate delays (bounded but unknown)
  - Ideal zero-delay wires

Source: [Sparso 06]
STG Synthesis

- Specification
- State graph
- State Graph with CSC
- Next-State functions
- Decomposed functions
- Gate netlist

Reachability analysis
State encoding
Boolean minimization
Logic decomposition
Technology mapping
Specification

Environment

Source: [Sparso 06]
State Graph

P0

\[ a^+ \quad c^- \quad b^+ \quad c^+ \]

P1

\[ a^- \quad d^- \quad d^+ \quad c^+ \]

(a, b, c, d)
Excitation Regions for Output Signal c

P0

ER1(c+)

ER1(c-)

ER2(c+)

QR1(c+)

b+

a+

b-

c-

d+

P1

0000

0010

0100

0110

1000

1010

1100

1110

1111

a+

b+

b+

c-

d+

a-

d-

c+
Quiescent Regions for Output Signal $c$
Next-State Functions
KV Diagram for c

<table>
<thead>
<tr>
<th>cd</th>
<th>ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td></td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>F</td>
</tr>
<tr>
<td>01</td>
<td>R</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0</td>
<td>R</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Atomic Complex Gate Implementation

Attention: Decomposition into simple gates can introduce hazards!

\[ c = d + a'b + bc \]
State-holding Gates Implementation

- Signals toggle between excitation and quiescent/stable regions
  - ER(c+) $\rightarrow$ QR(c+) $\rightarrow$ ER(c-) $\rightarrow$ QR(c-) etc.
- Implementation with SR-latches, C-gates or generalized C-gates possible

Source: [Sparso 06]
State-holding Gates

Set/Reset Functions

- $c = Set + c \cdot \overline{Reset}$
- $Set \cdot Reset = 0$

Set Function:
- must contain all states in ER($c+$)
- may contain states in QR($c+$)
- may contain not reachable states

Reset Function:
- must contain all states in ER($c-$)
- may contain states in QR($c-$)
- may contain not reachable states
**State-holding Gates Implementation**

Set function: \( c\text{-set} = d + a'b \)

<table>
<thead>
<tr>
<th>cd \ ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>F</td>
</tr>
<tr>
<td>01</td>
<td>R</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>R</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>X</td>
</tr>
</tbody>
</table>
Set function:
c-set = d + a'b

Reset function:
c-reset = b'

<table>
<thead>
<tr>
<th>cd</th>
<th>ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>F</td>
</tr>
<tr>
<td>00</td>
<td>R</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>R</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
State-holding Gates Implementation

Set function:  
\[ c\text{-set} = d + a'b \]

Reset function:  
\[ c\text{-reset} = b' \]

<table>
<thead>
<tr>
<th>cd</th>
<th>ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>R</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>R</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
State-holding Gates
Hazards

Diagram showing state transitions and hazard analysis for digital logic gates.
State-holding Gates
Monotonic Cover Constraint

- A cube (product term) may only be entered through ER states (monotonic cover or unique entry constraint)

Hazardous set function: c-set = d + a'b
State-holding Gates
Monotonic Cover Constraint

- A cube (product term) may only be entered through ER states (monotonic cover or unique entry constraint)

<table>
<thead>
<tr>
<th>cd</th>
<th>ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>R</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>R</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>R</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Fixed set function:
c-set = d + a'bc'
Example
VME Bus Controller

STG of Read Cycle
VME Bus Controller
CSC Conflict

(DSr, DTACK, LDTACK, LDS, D)
VME Bus Controller
CSC Conflict

\[ 10110 \]

\begin{align*}
\text{DSr}^+ & \quad \leftrightarrow \quad \text{DTACK}^- \\
\text{LDS}^+ & \quad \rightarrow \quad \text{LDTACK}^+ & \quad \rightarrow \quad \text{D}^+ & \quad \rightarrow \quad \text{DTACK}^+ & \quad \rightarrow \quad \text{DSr}^- & \quad \rightarrow \quad \text{D}^- \\
\text{LDTACK}^- & \quad \leftarrow \quad \text{LDS}^- \\
\text{DSr}^+ & \quad \leftrightarrow \quad \text{DTACK}^- \\
\text{LDS}^+ & \quad \rightarrow \quad \text{LDTACK}^+ & \quad \rightarrow \quad \text{D}^+ & \quad \rightarrow \quad \text{DTACK}^+ & \quad \rightarrow \quad \text{DSr}^- & \quad \rightarrow \quad \text{D}^- \\
\text{LDTACK}^- & \quad \leftarrow \quad \text{LDS}^- 
\end{align*}
Solution I: Remove conflict state by concurrency reduction
Solution I: Remove conflict state by concurrency reduction
Resolving CSC Conflict
Concurrency Reduction

- Concurrency reduction reflected by adding an arc to the STG specification.
- Introduces timing assumption (LDS- before DSr+)}
Resolving CSC Conflict
Adding State Signal

- **Solution II:** Inserting an internal state signal to make conflict states unique

![Diagram showing state transitions and signal states for resolving CSC conflicts](image)
Petrify

- Synthesis of speed independent control circuits from STG specifications
  - Simple text format for describing STGs
- Petrify can solve CSC problem
- Public domain tool
  - Developed at different universities
  - http://www.lsi.upc.edu/~jordicf/petrify/
Petrify - Example

.model cgate
.inputs a b
.outputs c
.graph
a+ c+
b+ c+
c+ a-
c+ b-
a- c-
b- c-
c- a+
c- b+
.marking { <c-, a+> <c-, b+> }
.end
Petrify

- Set of command-line tools
  - petrify: synthesis command
  - write_sg: derives state graph
  - draw_astg: draws STGs/state graphs

- Different circuit implementations
  - Complex gates (-cg)
  - Generalized C-elements (-gc)
  - Specific target library (-tm)
Summary

- Control logic essential part of asynchronous circuits
- PNs/STGs convenient for modeling control circuits
- STGs need to fulfill certain properties
  - Input-free choice, 1-bounded, CSC, etc.
- Synthesis from STGs to SI gate implementations possible
  - Tool available: Petrify