Chapter Overview

- HDLs allow specification of large systems.
- Graphs allow pictorial representation of small examples, and they are used by virtually every CAD algorithm.
- The chapter discusses the following types of graphs:
  - State machines
  - Petri-nets

Graph Basics

- A graph $G$ is composed of a finite nonempty set of vertices $V$ and a binary relation, $R (R \subseteq V \times V)$.
- Undirected graphs:
  - $R$ is an irreflexive symmetric relation.
  - Since $R$ is symmetric, $(u, v) \in R \Rightarrow (v, u) \in R$.
  - $E$ is the set of symmetric pairs, or edges (denoted $uv$).
- Directed graphs, or digraphs:
  - $R$ does not need to be either irreflexive or symmetric.
  - $E$ is the set of directed edges or arcs (denoted $(u, v)$).

A Simple Graph

A Simple Directed Graph

Additional Graph Definitions

- $|V|$ is called the order of $G$.
- $|E|$ is called the size of $G$.
- $V(G)$ and $E(G)$ are the vertex and edge sets for $G$.
- If $e = (u, v) \in E(G)$, $e$ joins $u$ and $v$.
- If $e = (u, v) \in E(G)$, $u$ and $v$ are incident with $e$.
- If $(u, v) \in E(G)$, $v$ is adjacent to $u$.
- If $(u, v) \notin E(G)$, $u$ and $v$ are nonadjacent vertices.
**Connected Graphs**

- **u-v path** is an alternating sequence of vertices and edges beginning with \( u \) and ending with \( v \).
- The length of a \( u-v \) path is the number of edges in the path.
- If there exists a \( u-v \) path, then \( v \) is reachable from \( u \).
- A \( u-v \) path is **simple** if it does not repeat any vertex.
- If for every pair of vertices \( u \) and \( v \) there exists a \( u-v \) path, the graph is connected.

**A Unconnected Graph**

![Unconnected Graph](image)

**Directed Acyclic Graphs**

- In a digraph, a \( u-v \) path forms a **cycle** if \( u = v \).
- If the \( u-v \) path excluding \( u \) is simple, then the cycle is **simple**.
- A cycle of length 1 is a **self-loop**.
- A digraph with no self-loops is **simple**.
- In an undirected graph, a \( u-v \) path is a cycle only if simple.
- A graph which contains no cycles is **acyclic**.
- An acyclic digraph is called a **directed acyclic graph** or **DAG**.

**A Cyclic Digraph**

![Cyclic Digraph](image)

**More Graph Properties**

- A digraph \( G \) is **strongly connected** if for every two distinct vertices \( u \) and \( v \), there exists a \( u-v \) path and a \( v-u \) path.
- A graph is **bipartite** if there exists a partition of \( V \) into two subsets \( V_1 \) and \( V_2 \) such that every edge of \( G \) joins a vertex of \( V_1 \) with \( V_2 \).
- A labeled graph is a triple \((V, R, L)\) in which \( L \) is a labeling function associated either to the set of vertices or edges.

**A Strongly Connected Digraph**

![Strongly Connected Digraph](image)
A Simple Labeled Directed Graph

v1
  a
  b
v2
v3
c  d
v4
e
v5

Chris J. Myers (Lecture 4: Graphs) Asynchronous Circuit Design

A Synchronous FSM

INPUTS

Comb. Logic

Register

OUTPUTS

STATE

CLOCK

Chris J. Myers (Lecture 4: Graphs) Asynchronous Circuit Design

An Asynchronous FSM

INPUTS

Comb. Logic

Delay

OUTPUTS

STATE

Chris J. Myers (Lecture 4: Graphs) Asynchronous Circuit Design

Finite State Machines

- \( I \) is the input alphabet;
- \( O \) is the output alphabet;
- \( S \) is the finite, non-empty set of states;
- \( S_0 \subseteq S \) is the set of initial (reset) states;
- \( \delta : S \times I \rightarrow S \) is the next-state function;
- \( \lambda : S \times I \rightarrow O \) is the output function for a Mealy machine (or \( \lambda : S \rightarrow O \) for a Moore machine).

Chris J. Myers (Lecture 4: Graphs) Asynchronous Circuit Design

Finite State Machine Diagrams

- FSM’s are often represented using a labeled digraph.
- The vertex set contains the states (i.e., \( V = S \)).
- The edge set contains the set of state transitions (i.e., \( (u, v) \in E \iff \exists i \in I \) s.t. \((u, i), v \) \in \( \delta \)).
- The labeling function is defined by next-state and output functions.
  - Each edge \((u, v)\) is labeled with \( i/o \) where \( i \in I \) and \( o \in O \) and \( ((u, i), v) \in \delta \) and \( ((u, i), o) \in \lambda \).

Chris J. Myers (Lecture 4: Graphs) Asynchronous Circuit Design

Passive/Active Shop

shop_PA_1: process
begin
  guard(req_wine,'1');  // winery calls
  assign(ack_wine,'1',1,3); // receives wine
  guard(req_wine,'0');  // req_wine reset
  assign(req_patron,'1',1,3); // call patron
  guard(ack_patron,'1');  // wine purchased
  assign(req_patron,'0',1,3); // reset req_patron
  guard(ack_patron,'0');  // ack_patron reset
  assign(ack_wine,'0',1,3); // reset ack_wine
end process;

Chris J. Myers (Lecture 4: Graphs) Asynchronous Circuit Design
Passive/Active Shop FSM

Burst-Mode State Machine

Burst-Mode State Machines

Input and Output Bursts

Maximal Set Property

Maximal Set Property
BM State Diagrams

- Not every BM state diagram represents a legal BM machine.
- If mislabeled with transitions that are not possible, it is impossible to define the in and out functions.
- There must be a strict alternation of rising and falling transitions on every input and output signal, across all paths.

Extended Burst-Mode

- BM machines require prescribed order: inputs change, outputs change, and state signals change.
- In extended burst-mode (XBM) state machines, this limitation is loosened a bit by the introduction of directed don't cares.
- These allow one to specify that an input change may or may not happen in a given input burst.
- BM machines also are unable to express conditional behavior.
- To support this type of behavior, XBM machines allow conditional input bursts.

Directed Don't Cares

Shop_PA_2: process
begin
  guard(req_wine,'1'); - winery calls
  assign(ack_wine,'1',1,3); - receives wine
  guard(req_wine,'0'); - req_wine reset
  assign(ack_wine = '0',1,3,req_patron,'1',1,3);
  guard(ack_patron,'1'); - wine purchased
  assign(req_patron,'0',1,3);
  guard(ack_patron,'0'); - reset req_patron
  guard(ack_patron,'0'); - ack_patron reset
end process;
Directed Don't Cares

- A transition is terminating when it is of the form \( t^+ \) or \( t^- \).
- A directed don't care transition is of the form \( t^* \).
- A compulsory transition is a terminating transition which is not preceded by a directed don't care transition.
- Each input burst must have at least one compulsory transition.

Modified Maximal Set Property

Conditional Input Bursts

A conditional input burst includes a regular input burst and a conditional clause.

A clause of the form \(<s^->\) indicates that the transition is only taken if \( s \) is low.

A clause of the form \(<s^+>\) indicates that the transition is only taken if \( s \) is high.

The signal in the conditional clause must be stable before every compulsory transition in the input burst.

Control PA_2: process
begin
  guard(req_wine,'1');
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(ack_wine,'1',1,3);
  guard(req_wine,'0');
  if (shelf = '0') then
    assign(ack_wine,'0',1,3,req_patron1,'1',1,3);
    guard(ack_patron1,'1');
    assign(req_patron1,'0',1,3);
    guard(ack_patron1,'0');
  end if;
  end process;
Conditional Input Bursts

\[
\begin{align*}
V & \subseteq V \times V \text{ is the set of edges (or transitions).} \\
I & = \{ x_1, \ldots, x_n \} \text{ is the set of inputs.} \\
O & = \{ z_1, \ldots, z_n \} \text{ is the set of outputs.} \\
C & = \{ c_1, \ldots, c \} \text{ is the set of conditional signals.} \\
V_0 & \subseteq V \text{ is the start state.} \\
in : V \rightarrow \{ 0, 1, * \}^m & \text{ defines } m \text{ inputs upon entry to each state.} \\
out : V \rightarrow \{ 0, 1 \}^n & \text{ defines } n \text{ outputs upon entry to each state.} \\
\text{cond} : E \rightarrow \{ 0, 1, * \} \text{ defines needed conditional inputs.}
\end{align*}
\]
**Petri-Nets**

- A Petri-net is a bipartite digraph.
- The vertex set is partitioned into two disjoint subsets:
  - \( P \) is the set of places.
  - \( T \) is the set of transitions.
- The set of arcs, \( F \), is composed of pairs where one element is from \( P \) and the other is from \( T \) (i.e., \( F \subseteq (P \times T) \cup (T \times P) \)).
- A Petri-net is \( \langle P, T, F, M_0 \rangle \) where \( M_0 \) is the initial marking.

**Presets and Postsets**

- The preset of a transition \( t \in T \) (denoted \( •t \)) is the set of places connected to \( t \) (i.e., \( •t = \{ p \in P | (p, t) \in F \} \)).
- The postset of a transition \( t \in T \) (denoted \( t^• \)) is the set of places \( t \) is connected to (i.e., \( t^• = \{ p \in P | (t, p) \in F \} \)).
- The preset of a place \( p \in P \) (denoted \( •p \)) is the set of transitions connected to \( p \) (i.e., \( •p = \{ t \in T | (p, t) \in F \} \)).
- The postset of a place \( p \in P \) (denoted \( p^• \)) is the set of transitions \( p \) is connected to (i.e., \( p^• = \{ t \in T | (p, t) \in F \} \)).

**Markings**

- A marking, \( M \), for a Petri net is a function that maps places to natural numbers (i.e., \( M : P \rightarrow \mathbb{N} \)).
- Markings can be added or subtracted using vector arithmetic.
- They can also be compared:
  \[ M \geq M' \iff \forall p \in P . \ M(p) \geq M'(p) \]
- For a set of places, \( A \subseteq P \), \( CA \) denotes the characteristic marking of \( A \):
  \[ CA(p) = \begin{cases} 1 & \text{if } p \in A \\ 0 & \text{else} \end{cases} \]

**Transition Firings**

- A transition \( t \) is enabled under the marking \( M \) if \( M \geq CA_t \).
- In other words, \( M(p) \geq 1 \) for each \( p \in •t \).
- The firing transforms the marking as follows (denoted \( M(t)M' \)):
  \[ M' = M - CA_t + CA_{t^•} \]
- When a transition \( t \) fires, a token is removed from each place in its preset, and a token is added to each place in its postset.
Reachable Markings

- Firing of a transition transforms the marking of the Petri net into a new marking.
- A sequence of transition firings \((\sigma = t_1, t_2, \ldots, t_n)\) produces a sequence of markings \((M_0, M_1, \ldots, M_n)\).
- If such a firing sequence exists, we say that the marking \(M_n\) is reachable from \(M_0\) by \(\sigma\) (denoted \(M_0[\sigma]M_n\)).
- We denote the set of all markings reachable from a given marking by \([M]\).

Example Firing Sequence

- A sequence of transitions produces a sequence of markings.
- Michael Jackson's album includes songs like "Thriller" and "Bad."
Liveness Categories

- Different liveness categories can be determined more easily.
- In particular, a transition $t$ for a given Petri net is said to be:
  - **dead** ($L_0$-live) if there does not exist a firing sequence in which $t$ can be fired.
  - **$L_1$-live** (potentially firable) if there exists at least one firing sequence in which $t$ can be fired.
  - **$L_2$-live** if $t$ can be fired at least $k$ times.
  - **$L_3$-live** if $t$ can be fired infinitely often in some firing sequence.
  - **$L_4$-live or live** if $t$ is $L_1$-live in every marking reachable from the initial marking.

- A Petri net is $L_k$-live if every transition in the net is $L_k$-live.

Reachability Graph

- When a Petri net is bounded, the number of reachable markings is finite, and a reachability graph (RG) can be found.
- In an RG, the vertices, $\Phi$, are the markings and the edges, $\Gamma$, are the possible transition firings between two markings.
- For safe Petri nets, vertices in RG are labeled with the subset of places included in the marking.
- The edges are labeled with the transition that fires to move the Petri net from one marking to the next.

Example Reachability Graph

```
find_RG(Petri net \langle P, T, F, M_0 \rangle)
M = M_0; T_e = \{t \in T | M \geq C \cdot t\}; \Phi = \{M\}; \Gamma = \emptyset;
done = false;
while (\neg done)
  t = select(T_e);
  if (T_e - \{t\} \neq \emptyset) then push(M, T_e - \{t\});
  M' = M - C \cdot t + C \cdot t;
  if (M' \in \Phi) then
    \Phi = \Phi \cup \{M'\}; \Gamma = \Gamma \cup \{(M, M')\};
    M = M'; T_e = \{t \in T | M \geq C \cdot t\};
  else
    \Gamma = \Gamma \cup \{(M, M')\};
  if (stack is not empty) then (M, T_e) = pop();
  else done = true;
return (\Phi, \Gamma);
```

Safe Example

```
producing receive sending consuming
p1
p3
p5
p2
p4
p6
```

Example Reachability Graph

```
(p1,p4,p5)
produce consume
(p2,p4,p6)
produce
(p3,p3,p5)
consume produce
(p3,p3,p6)
send
(p1,p4,p5)
produce consume
(p2,p4,p6)
produce
(p1,p1,p6)
consume produce
```
**Concurrency, Conflict, and Confusion**

- Two transitions $t_1$ and $t_2$ are concurrent when there exists markings where both are enabled and can fire in either order.
- Two transitions, $t_1$ and $t_2$, are in conflict when the firing of one disables the firing of the other.
- When concurrency and conflict are mixed, we get confusion.

**State Machines and Marked Graphs**

- A Petri-net is a state machine if and only if every transition has exactly one place in its preset and one place in its postset.
  \[
  \forall t \in T : |\bullet t| = |t \bullet| = 1
  \]
- State machines do not allow concurrency, but do allow conflict.
- A Petri-net is a marked graph if and only if every place has exactly one transition in its preset and one in its postset.
  \[
  \forall p \in P : |\bullet p| = |p \bullet| = 1
  \]
- Marked graphs do not allow conflict, but do allow concurrency.

**Free Choice Nets**

- A Petri-net is free choice if and only if every pair of transitions that share a common place in their preset have only a single place in their preset.
  \[
  \forall t, t' \in T, t \neq t' : t \cap t' \neq \emptyset \Rightarrow |\bullet t| = |\bullet t'| = 1
  \]
  \[
  \forall p, p' \in P, p \neq p' : p \cap p' \neq \emptyset \Rightarrow |p \bullet| = |p' \bullet| = 1
  \]
  \[
  \forall p \in P, \forall t \in T : (p, t) \in F \Rightarrow p^\bullet = \{t\} \cup t = \{p\}
  \]
- Free choice nets allow concurrency and conflict, but do allow confusion.
Extended Free Choice Nets

- A Petri net is an extended free choice net if and only if every pair of places that share common transitions in their postset have exactly the same transitions in their postset.

\[ \forall p, p' \in P . \ p \cap p' \neq \emptyset \Rightarrow p = p' \]

- Extended free-choice nets also allow concurrency and conflict, but they do not allow confusion.

Asymmetric Choice Nets

- A Petri net is an asymmetric choice net if and only if for every pair of places that share common transitions in their postset, one has a subset of the transitions of the other.

\[ \forall p, p' \in P . \ p \cap p' \neq \emptyset \Rightarrow p \subseteq p' \lor p' \subseteq p \]

- Asymmetric choice nets allow asymmetric confusion but not symmetric confusion.

Checking Safety and Liveness

- It is possible to check safety and liveness for certain restricted classes of Petri nets using the theorems given below.

**Theorem 4.1** A state machine is live and safe iff it is strongly connected and \( M_0 \) has exactly one token.

**Theorem 4.2** (Commoner, 1971) A marked graph is live and safe iff it is strongly connected and \( M_0 \) places exactly one token on each simple cycle.
### Siphons and Traps

- A **siphon** is a nonempty subset of places, $S$, in which every transition having a postset place in $S$ also has a preset place in $S$ (i.e., $S \subseteq S\downarrow$).
- If in some marking no place in $S$ has a token, then in all future markings, no place in $S$ will ever have a token.
- A **trap** is a nonempty subset of places, $Q$, in which every transition having a preset place in $Q$ also has a postset place in $Q$ (i.e., $Q\uparrow \subseteq Q\downarrow$).
- If in some marking some place in $Q$ has a token, then in all future markings some place in $Q$ will have a token.

### Checking Liveness

**Theorem 4.3 (Hack, 1972)** A free-choice net, $N$, is live iff every siphon in $N$ contains a marked trap.

**Theorem 4.4 (Commoner, 1972)** An asymmetric choice net $N$ is live if (but not only if) every siphon in $N$ contains a marked trap.

### Example Siphon and Trap

![Example Siphon and Trap](image)

### Marked Graph Components

- A **marked graph component** of a net, $N$, is a subnet in which each place has at most one transition in its preset and one transition in its postset and is generated by these transitions.
- The net generated by a set of transitions includes these transitions, all places in their preset and postset, and all connecting arcs.
- A net $N$ is said to be covered by a set of MG-components when the set of components includes all places, transitions, and arcs from $N$.

**Theorem 4.6** If $N$ is a live and safe free-choice net then $N$ is covered by strongly connected MG-components.

### State Machine Components

- A **state machine component** of a net, $N$, is a subnet in which each transition has at most one place in its preset and one place in its postset.
- The net generated by a set of places includes these places, all transitions in their preset and postset, and all connecting arcs.
- A net $N$ is said to be covered by a set of SM-components when the set of components includes all places, transitions, and arcs from $N$.

**Theorem 4.5 ( Hack, 1972)** A live free-choice net, $N$, is safe iff $N$ is covered by strongly connected SM-components each of which has exactly one token in $M_0$.

### Signal Transition Graphs (STG)

- To use a Petri net to model asynchronous circuits, must relate transitions to events on signal wires.
- Several variants of Petri nets accomplish this: $M$-nets, $I$-nets, and change diagrams.
- A signal transition graph (STG) is a labeled safe Petri net which is modeled by $(P, T, F, M_0, N, s_0, \lambda_T)$, where:
  - $N = I \cup O$ is the set of signals where $I$ is the set of input signals and $O$ is the set of output signals.
  - $s_0$ is the initial value for each signal in the initial state.
  - $\lambda_T : T \rightarrow N \times \{+, -\}$ is the transition labeling function.
- Each transition is labeled with either a rising transition, $s+$, or falling transition, $s-$.
- A STG imposes explicit restrictions on the environment.
STG Restrictions

- STGs are often restricted to a synthesizable subset.
- Synthesis methods often restrict the STG to be live and safe.
- Some synthesis methods require STGs to be persistent.
- A STG is persistent if for all \( a^+ \rightarrow b^- \), there exist other arcs that ensure that \( b^+ \) fires before the opposite transition of \( a^- \).
- Other methods require single-cycle transitions.
- A STG has single-cycle transitions if each signal name appears in exactly one rising and one falling transition.
- None of these restrictions is actually a necessary requirement for a circuit implementation to exist.
- These restrictions can simplify the synthesis algorithms.

Liveness

Safety

Persistency

Single-Cycle Transitions
State Graphs (SG)

To design a circuit from an STG, must find its state graph.
A SG is modeled by the tuple \((S, \delta, \lambda_S)\).
- \(S\) is the set of states.
- \(\delta \subseteq S \times T \times S\) is the set of state transitions.
- \(\lambda_S : S \to \{0, 1\}\) is the state labeling function.
- Each state \(s\) is labeled with a vector \((s(0), s(1), \ldots, s(n))\), where \(s(i)\) is either 0 or 1, indicating value returned by \(\lambda_S\).
- We use \(s(i)\) interchangeably with \(\lambda_S(s)(i)\).

Implied State

- If in \(s_i\), there exists a transition on signal \(u_i\) to \(s_j\) [i.e., \(\exists (s_i, t, s_j) \in \delta : \lambda_T(t) = u_i + \lambda_T(t) = u_i - \), then \(u_i\) is excited.
- Otherwise, the signal \(u_i\) is in equilibrium.

The value each signal is tending to is called its implied value.
- If the signal is excited, the implied value of \(u_i\) is \(\overline{s_i(t)}\).
- If the signal is in equilibrium, the implied value of \(u_i\) is \(s_i(t)\).
- The implied state, \(s_i'\) is labeled with a binary vector \((s_i'(0), s_i'(1), \ldots, s_i'(n))\) of the implied values.
- The function \(X : S \to 2^T\) returns the set of excited signals in a given state [i.e., \(X(s) = \{u_i \in S | s_i(t) \neq s_i(t)\}\)].
- When \(u_i \in X(s)\) and \(s_i(t) = 0\), \(s_i(t)\) in SG is marked with “R”.
- When \(u_i \in X(s)\) and \(s_i(t) = 1\), \(s_i(t)\) in SG is marked with “F”.

Consistent State Assignment

- A well-formed SG, must have a consistent state assignment.
- A SG has a consistent state assignment if for each state transition \((s_i, t, s_j) \in \delta\) exactly one signal changes value, and its value is consistent with the transition.
- \(\forall (s_i, t, s_j) \in \delta . \forall u \in N . (\lambda_T(t) \neq u \land s_i(u) = s_j(u)) \lor (\lambda_T(t) = u \land s_i(u) = 0 \land s_j(u) = 1) \lor (\lambda_T(t) = u \land s_i(u) = 1 \land s_j(u) = 0)\)

where “\(\lor\)" represents either “+" or “−".
- A STG produces a SG with a consistent state assignment if in any firing sequence, transitions of a signal strictly alternate between +’s and −’s.
Consistent State Assignment

Unique State Code

A SG has a unique state assignment (USC) if no two different states (i.e., markings) have identical values for all signals [i.e., \( \forall s, s' \in S, s \neq s', \lambda(s) \neq \lambda(s') \)].

Some synthesis methods are restricted to STGs that produce SGs with USC.

Reshuffled Passive/Lazy-Active Wine Shop

Shop_PA_lazy_active: process
begin
  guard(req_wine,'1'); - winery calls
  assign(ack_wine,'1',1,3); - receives wine
  guard(ack_patron,'0'); - ack_patron reset
  assign(req_patron,'1',1,3); - call patron
  guard(req_wine,'0'); - req_wine reset
  assign(ack_wine,'0',1,3); - reset ack_wine
  guard(ack_patron,'1'); - wine purchased
  assign(req_patron,'0',1,3); - reset req_patron
end process;

Wine Shop with Two Patrons

Shop_PA_2: process
begin
  guard(req_wine,'1');
  shelf <= bottle after delay(2,4);
  wait for delay(5,10);
  assign(ack_wine,'1',1,3);
  guard(req_wine,'0');
  if (shelf = '0') then
    assign(ack_wine,'0',1,3,req_patron1,'1',1,3);
    guard(ack_patron1,'1');
    assign(req_patron1,'0',1,3);
    guard(ack_patron1,'0');
Wine Shop with Two Patrons

```plaintext
elsif (shelf = '1')
    assign(ack_wine,'0',1,3,req_patron2,'1',1,3);
    guard(ack_patron2,'1');
    assign(req_patron2,'0',1,3);
    guard(ack_patron2,'0');
end if
end process;
```

Labeled Petri nets

- AFSMs cannot model arbitrary concurrency.
- Petri-nets have difficulty to express signal levels.
- **Labeled Petri nets** are a hybrid graphical representation method which are both capable of modelling arbitrary concurrency and signal levels.

Semantics

- A transition \( t \in T \) is enabled to fire when its present (\( \bullet t \)) is marked and its enabling condition (\( En(t) \)) evaluates to true in the current state.
- Once a transition is enabled it fires sometime between the lower and upper bound associated with its delay assignment (\( D(t) \)).
- When a transition fires, the marking is updated and the Boolean assignments associated with the transition (\( BA(t) \)) are executed.

Each transition \( t \in T \) has the following labels:
- \( En: T \rightarrow \mathcal{P} \) - the enabling condition;
- \( D: T \rightarrow \mathbb{Q} \times (\mathbb{Q} \cup \{\infty\}) \) - the delay assignment; and
- \( BA: T \rightarrow 2^{\mathbb{B} \times \mathcal{P}} \) - set of Boolean assignments.

The language for the \( \mathcal{P} \) is defined as follows:

\[
\phi ::= \text{true} | \text{false} | b | \neg \phi | \phi \land \phi | \phi \lor \phi
\]
LPN for a C-Element

LPN for Wine Shop with Two Patrons

Summary

- Finite state machines (AFSMs, BM, and XBM).
- Petri-nets, STGs, and LPNs.