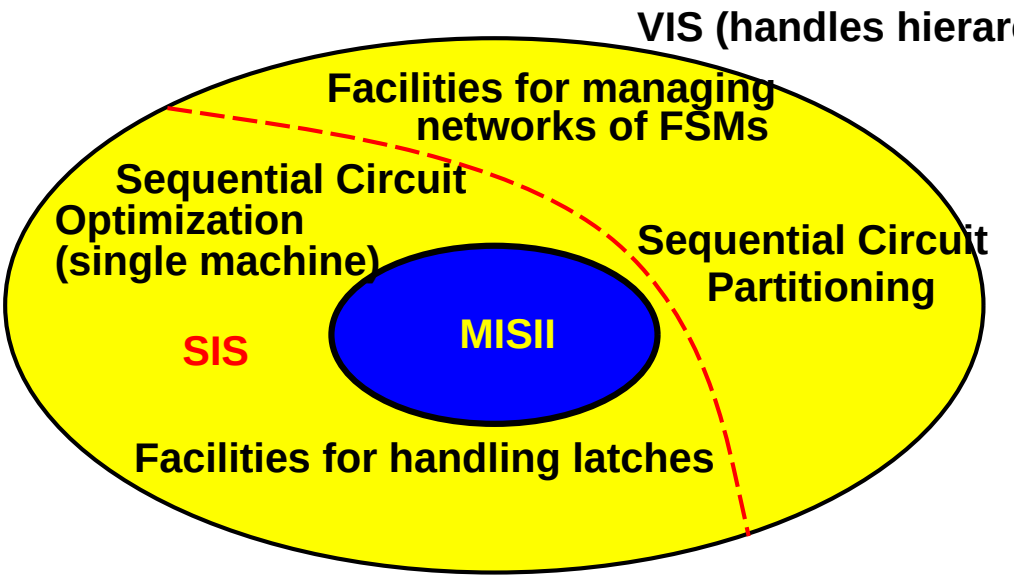
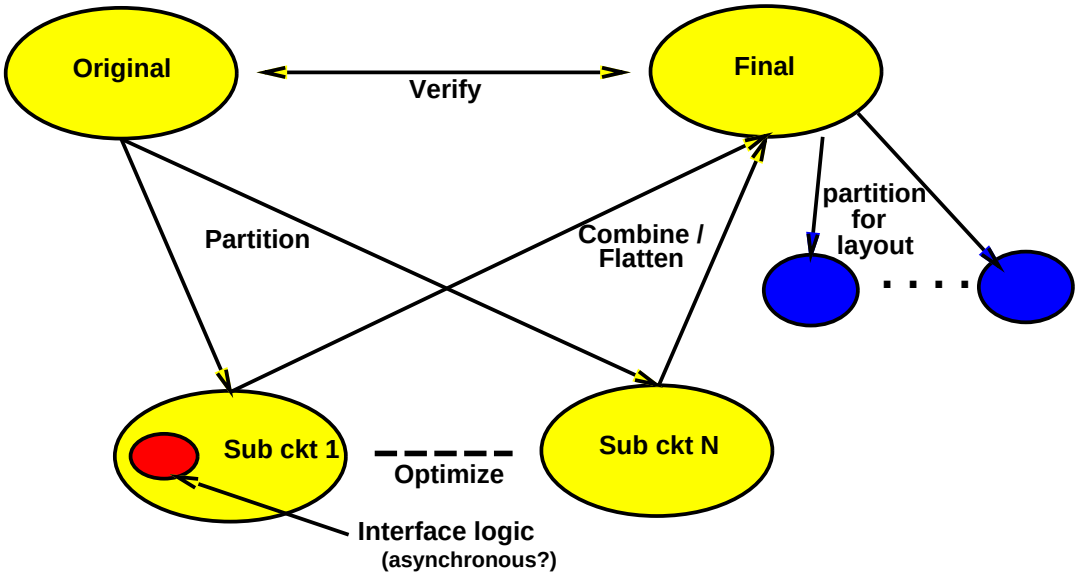
FSM Introduction

History: Combinational Logic $\rightarrow$ single FSM $\rightarrow$ Hierarchy of FSM's.





What are Combinational Circuits?

Definition: A circuit is combinational if it computes a function which depends only on the inputs applied to the circuit; for every input value, there is a unique output value.

- Circuits with an acyclic underlying topology are combinational.

- Cyclic circuits can be combinational, in fact, there are combinational circuits whose minimal form must have cycles [Kautz 1970].

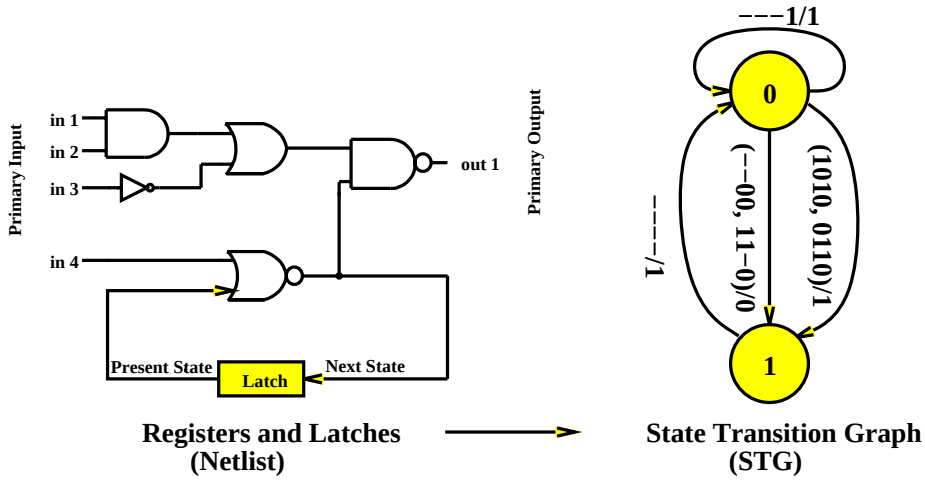
- Recent work on checking if circuit combinational [Malik'94, Shiple'95]. These are based on X-valued simulation.

What are Sequential Circuits?

- Some sequential circuits have memory elements. Synchronous circuits have clocked latches. Asynchronous circuits may or may not have latches (e.g. C-elements), but these are not clocked.

- Feedback (cyclic) is a necessary, but not sufficient condition for a circuit to be sequential.

- Synthesis of sequential circuits is not as well developed as combinational. Sequential synthesis techniques not really used in commercial software.



The above circuit is sequential since output depends on the state and input.

Example - Highway Light (Verilog)

```
module hwy_control(clk, car_present, enable_hwy,
        short_timer, long_timer, hwy_light,
        hwy_start_timer, enable_farm);
input clk,car_present,enable_hwy,short_timer,
       long_timer;
output hwy_light,hwy_start_timer,enable_farm;
boolean wire car_present;
wire short_timer, long_timer, hwy_start_timer,
       enable_farm, enable_hwy;
color reg hwy_light;

initial hwy_light = GREEN;

assign hwy_start_timer = (((hwy_light == GREEN)
        && ((car_present  == YES) && long_timer))
                    ||
                (hwy_light == RED) && enable_hwy);
assign enable_farm=((hwy_light==YELLOW)&&short_timer);
always @(posedge clk) begin
        case (hwy_light)
        GREEN: if ((car_present == YES) &&
                long_timer) hwy_light = YELLOW;
        YELLOW: if (short_timer) hwy_light = RED;
        RED: if (enable_hwy) hwy_light = GREEN;
        endcase
end
endmodule
```

Finite State Machines

Finite State Machines in STG or transition relation form are a behavioral view of sequential circuits. They describe the transitional behavior of these circuits. They can distinguish among a **finite** number of classes of **input histories**: these classes are the *internal states* of the machine.

Moore Machine: is a quintuple

$$M = (S, I, O, \delta, \lambda)$$

S : finite non-empty set of states
I : finite non-empty set of inputs
O : finite non-empty set of outputs
$\delta : S \times I \mapsto S$ transition (or next state) function
$\lambda : S \mapsto O$ output function

Mealy Machine: $M = (S, I, O, \delta, \lambda)$ but

$\lambda : S \times I \mapsto O$

For digital circuits, typically $I = \{0,1\}^m$ and $O = \{0,1\}^n$.

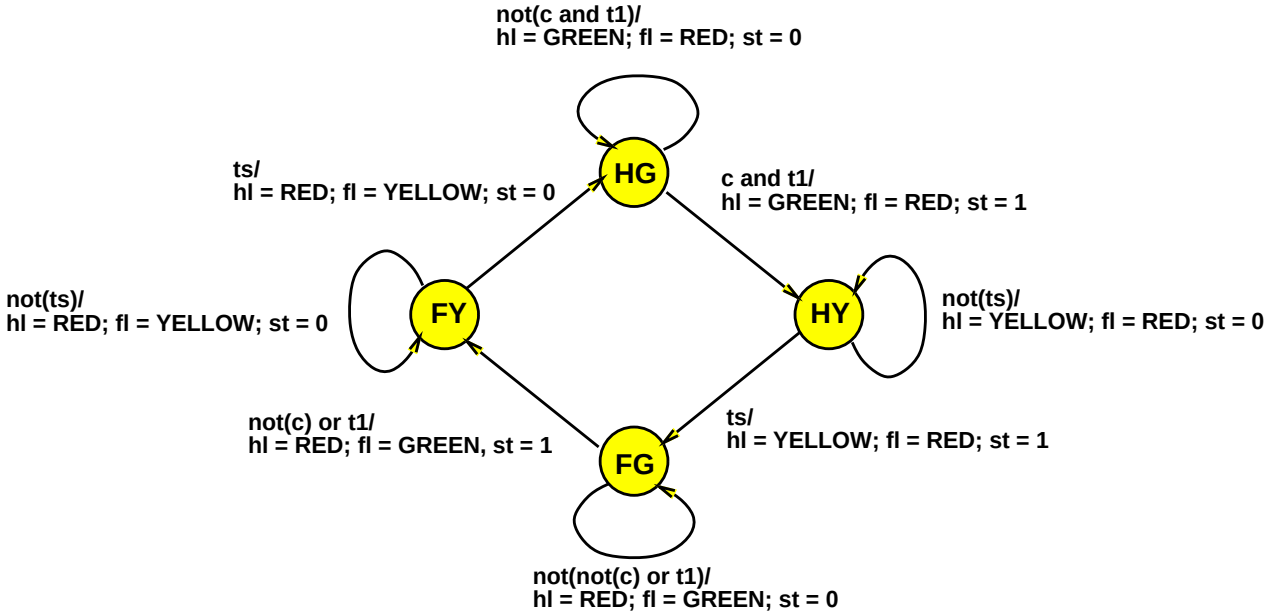
In addition certain states may be classified as *reset or initial states*.

Automata are similar to FSM's, however they do not produce any outputs, they just *accept* input sequences (*accepting set of states is given*).

Representing State Machines

State Transition Graphs and Tables

Example: Traffic Light Controller - Mealy machine



State Transition Graph: Example

PS	IN	NS	OUT
HG	not(c and t1)	HG	hl = GREEN; fl = RED; st = 0
HG	c and t1	HY	hl = GREEN; fl = RED; st = 1
HY	not(ts)	HY	hl = YELLOW; fl = RED; st = 0
HY	ts	FG	hl = YELLOW; fl = RED; st = 1
FG	not(not(c) or t1)	FG	hl = RED; fl = GREEN; st = 0
FG	not(c) or t1	FY	hl = RED; fl = GREEN; st = 1
FY	not(ts)	FY	hl = RED; fl = YELLOW; st = 0
FY	ts	HG	hl = RED; fl = YELLOW; st = 1

State Transition Table: Example

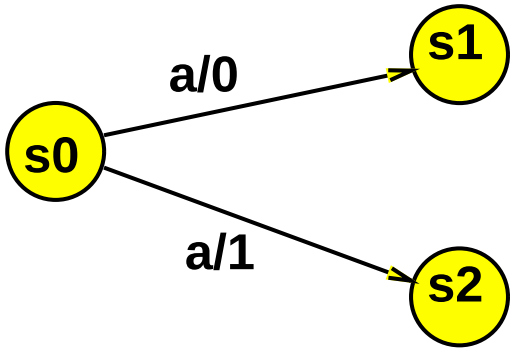
Representing State Machines

- State Transition Graphs and State Transition Tables are similar; the first is graphical, the second is tabular.

- In this example the edges (transitions) are labeled with general logic functions (predicates) of the inputs.

- Traditionally minterms or cubes have been used for the transitions (e.g. KISS format), especially for tables, since used as input to two-level minimizers. Minterms need the most edges and arbitrary logic functions (predicates) the least.

Non-Determinism and Incomplete Specification



- In automata theory, **non-determinism** is associated with many transitions; from a given current state and under the same input conditions we may go to different states and have different outputs. Each behavior is considered valid. Nondeterminism provides a **compact** way to describe a set of valid behaviors.

- In classical sequential function theory, transition functions and output functions can be **incompletely specified** (i.e. the functions can have don't cares), i.e. defined only on a proper subset of their input space. Where it is undefined, we consider it to allow any behavior. This also describes a **set** of valid behaviors.

Non-Determinism and Incomplete Specification

Given an input and present state:

- **Nondeterminism:** some next states and outputs are ruled out. Result is subset of next states and outputs admissible for a transition.

- **Don't cares:** all next states and outputs are allowed. These may be because the given state can't be reached, so will never occur, or the state is a binary code not used during state assignment.

- **Incomplete transition structure:** It may be that no next state is allowed. If this is because that input will never occur at that state we need to "complete" the description by adding transitions to all states and allowing all outputs. On the other hand, we may want the machine to do nothing (e.g. as an automaton). Sometimes we "complete" the transition structure by adding a **dummy state** and calling it a non-accepting state.

All describe a set of behaviors. These are used to describe flexibility for the implementation during synthesis, and to describe a subset of acceptable behaviors.

Non-Determinism and Incomplete Specification

- Optimization tools for logic synthesis and verification exploit in various fashions incomplete specification to achieve optimization objectives.

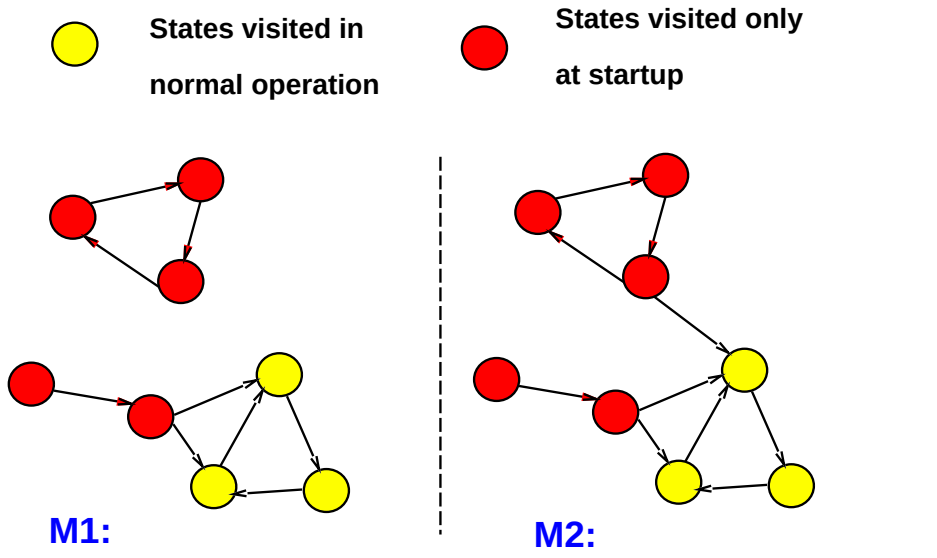
  More recently, methods to exploit flexibility given by non-determinism have been devised [Kim and Newborn, Somenzi, Wang, Watanabe, Kam&Villa]

- At the implementation level, only one of the possible next states and outputs is chosen (complete specification).

Initializing Sequences

Reference: [C. Pixley, TCAD Dec. 1992]

Q: How many states does a circuit (implementation) with n memory elements have?
A: 2^n, one for each possible vector of values of these memory elements. Must assume on power up, that any of the 2^n states is allowed.



The set of states of normal operation forms a strongly connected component.

Initializing Sequence: A sequence of input vectors that gets the machine to an equivalence class of known reset states.

May be implemented using a single reset signal.

Pixley: *If an aligning sequence exists for each state pair, then an initializing sequence exists.*

FSM Extraction

Explicit/Semi-Implicit Extraction of all Transitions

Method 1:
Reference: [Devadas-Ma-Newton88]

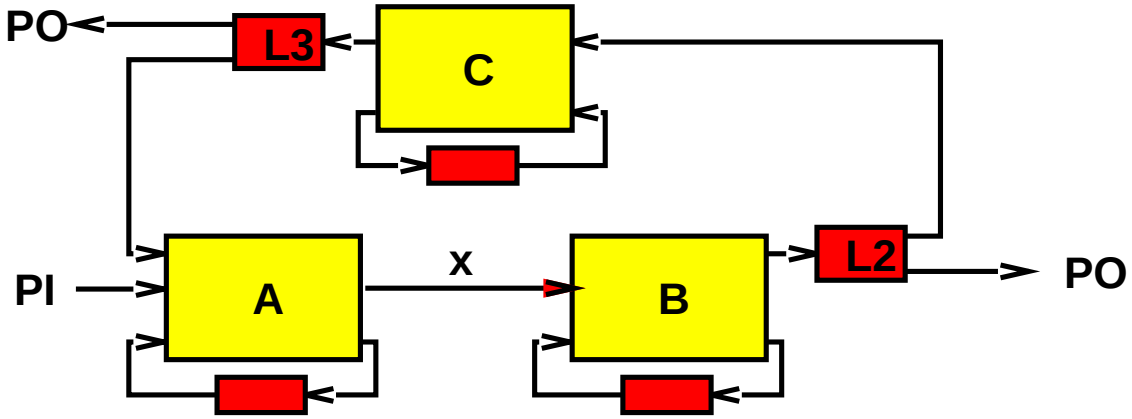
Visit states starting from the reset states (in breadth-first-order).

```
extract(C) {    /* C is the given circuit  */
    st_table =  { };
    list =  { };
    foreach(s in reset_states)
        add_list(list, s);
    while((ps = next_unvisited(list)) != NIL) {
      /* iterate till all states have been visited */
        while([(in, ns, out) <= generate_ns(ps)] != NIL) {
          /* generate transitions from ps one by one */
            st_table = st_table + {(in, ps, ns, out)};
            if(! in_list(list, ns)) add_list(list, ns);
        }
    mark_visited(list,ps);
    }
    return(st_table);
}
```

Of course, could do this in DFS order too, but see next slide.

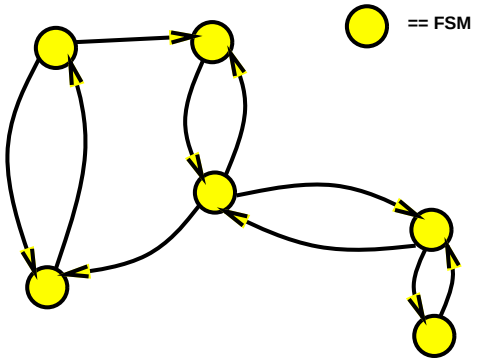
Interconnected FSMs - FSM Networks

- Natural way of describing complex systems (hierarchy, decomposition). Naturally extracted from HDL's with modules or sub-processes.



- Interconnected FSMs $\equiv$ Single product machine (similar to flattening in boolean circuits)

- Directed Graph - Each node an FSM. Arcs are variables used for communication.



Similar to Boolean network, possibly cyclic.