

ECE 587 – Hardware/Software Co-Design

Lecture 05 State-Based Models I

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

January 26, 2026

Models of Computation

Finite State Machine

Examples

Reading Assignment

- ▶ This lecture: 3.1, 3.1.2
- ▶ Next lecture: 3.1.2

Models of Computation

Finite State Machine

Examples

Models of Computation

- ▶ Any non-trivial functionality must involve some kind of computation.
- ▶ It is beneficial to specify the functionality just at the abstraction level of the computation.
 - ▶ It's intuitive.
 - ▶ Computations are behavioral. No implementation detail is necessary.
 - ▶ Computations are based on mathematics. There may exist tools to automate the remaining design process.
- ▶ Models of Computation (MoCs)
 - ▶ Serve as basis to reason about computation/constraints
 - ▶ Utilize formal language, e.g. certain kind of mathematics
 - ▶ May have different supported features, complexity, and expressive power.

- ▶ MoCs define computations by specifying when to perform operations.
 - ▶ The time here is not absolute time but relative ordering.
 - ▶ So ultimately it depends on how synchronizations are employed.
- ▶ Fully synchronized model: Finite State Machine
- ▶ Fully ordered without synchronization: Sequential Programs
- ▶ No synchronization at all: Dataflow
- ▶ We will first focus on FSM and move to other models in the next few weeks.

Models of Computation

Finite State Machine

Examples

Finite-State Machine (FSM)

$$\langle S, I, O, f, h \rangle$$

- ▶ Set of states S
- ▶ Set of input symbols I
- ▶ Set of output symbols O
- ▶ Next-state function $f : S \times I \rightarrow S$
- ▶ Output function $h : S \times I \rightarrow O$
- ▶ Some systems may specify initial states and/or final states

What is *not* specified?

- ▶ Encoding of states and input/output symbols in HW/SW
 - ▶ This condition will sometimes be relaxed so one can handle extremely large systems.
- ▶ Implementation of f and h in HW/SW

Representations of FSM

- ▶ Graph representation
 - ▶ States as vertices
 - ▶ State transitions as edges (annotated with inputs/outputs)
 - ▶ Intuitive, but if there are too many possible states, it becomes unmanageable.
- ▶ Functional representation
 - ▶ If one can efficiently specify f and h , then the FSM can be simulated from any initial state and a trace of inputs, fulfilling most computational tasks.
 - ▶ Can handle extremely large systems

Encodings

- ▶ Since a FSM has a finite number of possible states, one can represent, or *encode*, a state using a fixed number of bits.
 - ▶ E.g. if there are 16 possible states, a 4-bit encoding can be applied.
- ▶ Similarly you can encode inputs and outputs.
- ▶ Under such encodings, the functions f and h become boolean functions.

FSM vs. Register Transfer Level (RTL)

- ▶ That's exactly how RTL is defined.
 - ▶ Just change the state bits to registers
- ▶ The key here is encoding.
 - ▶ Encoding enables us to specify extremely large FSMs.
 - ▶ Different encodings may lead to specifications with different complexity, though for system design we prefer to use the most intuitive one.
- ▶ We will still distinguish functional representations of FSM from RTL as they have different purposes.
 - ▶ Though mathematically there is no difference.

Implement FSMs

- ▶ Hardware: as Synchronous Circuits
 - ▶ Utilize the connection between functional representation and RTL
 - ▶ Exactly one state transition happens per clock cycle.
 - ▶ High speed/low power/energy consumption
 - ▶ Usually known as cycle-accurate behavior
- ▶ Software: follow either graph or functional representations
 - ▶ Tedious, better to have tools to generate code
 - ▶ Not efficient in both time and power
 - ▶ But is a very powerful architecture to build complex software that needs to react to external events, e.g. networking and graphical user interface.

Outline

Models of Computation

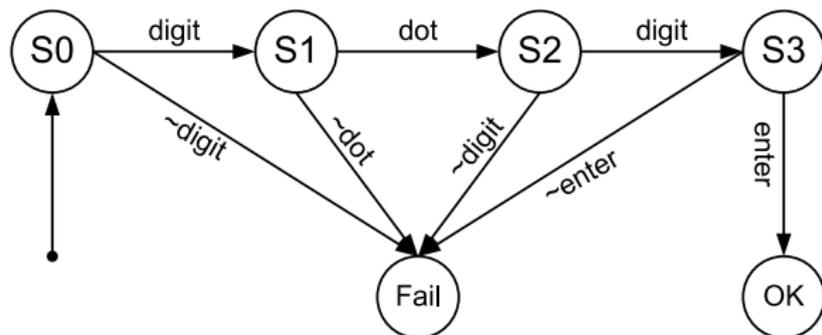
Finite State Machine

Examples

Input Validation

- ▶ Consider an application that requires to validate user inputted numbers
 - ▶ Assume the input is a character string
 - ▶ End of string must be `enter`.
- ▶ A valid integer
 - ▶ If the first character is not a digit, then it must be either `+` or `-`.
 - ▶ Except the first character and the ending `enter`, all characters are digits.
 - ▶ The most significant digit must not be 0.
 - ▶ The integer may contain arbitrary number of digits.
- ▶ Additional tasks
 - ▶ Deal with floating-point numbers
 - ▶ Extract the number during validation
 - ▶ Implement the designs in a programming language.
- ▶ How to approach this or similar problems?

A Simple FSM



- ▶ How does it work?
 - ▶ Starting from S0
 - ▶ Process exactly one character per transition.
- ▶ This simple example accepts numbers like 1.2, 4.5, but not 11 or 1.21.

A More Complex FSM

- ▶ Build a FSM to recognize integers.
- ▶ Extend it to handle floating-point numbers.

Extract Numbers

- ▶ Focus on integers but make it easy to extend our solution to floating-point numbers etc.

Software Implementation

```
enum {S0, S1, ..., OK, FAIL};
int state = S0, sign = 1, num = 0;
while ((state != OK) && (state != FAIL)) {
    int next_state = state; // assume state remain the same by default
    int ch = read_one_input();
    if (state == S0) {
        if (ch == '-') {
            next_state = S1; sign = -1;
        } else if (isdigit(ch)) {
            next_state = S1; num = ch-'0';
        } ...
    } else if (state == S1) {
        } ...
    state = next_state;
}
num *= sign;
```

- ▶ Make use of a single loop to drive the state transitions.
- ▶ Use two levels of branches to handle combinations of current state and input.
- ▶ It can handle *any* FSM no matter how complicated it is.

Discussions

- ▶ From the FSM model, it will be much easier for the designers to utilize tools at hand to implement the validation as either hardware or software.
- ▶ Such problems are special cases of *Regular Expressions*.
 - ▶ It is used almost everywhere when text is processed.
 - ▶ Many places require to run it very efficiently, e.g. to filter certain information from the network at realtime.
- ▶ Regular expressions can be modeled by a special kind of FSMs called nondeterministic FSM.
 - ▶ There is a mapping from graph representation of nondeterministic FSM to RTL, which enable one to implement it quite efficiently in hardware.
 - ▶ The challenge in hardware implementation is reconfigurability without much overhead.
 - ▶ Software implementations are based on the same idea but are much more awkward.