# Available as undergraduate Technical Elective

# ECE-548, SEQUENTIAL MACHINE THEORY

## INSTRUCTOR: DR. K. HINTZ

*If you don't believe that hardware and software are the same thing, then you need to take this course.*

## PRACTICAL AND USEFUL RESULT

*Every computer is comprised of 5 parts, one of which is the control unit. The control unit is most commonly implemented as an FSM. The material you will learn in this course enables you to find the* ***provably smallest FSM*** *that will implement the control unit. This is important from the point of view of execution speed, power consumption (battery life), and integrated circuit layout size.*

## How many times have you heard someone say that "You yo*u can do that in hardware but you can't do it in software,* or *You can do that in software but not in hardware.*

The point is that any digital computation that you can do in software can equivalently be done in hardware and *vice-versa*. The only difference is the *amount of hardware* and the *speed of execution*. The questions these days revolve around which part of the solution to do in hardware and which part to do in software. That decision depends on the intended use of the solution.

## First Half of the Course: Formal Languages

The purpose of this course is to show that formal computer languages as defined by the Chomsky hierarchy of languages can be implemented as language recognizers by different classes of machines. We begin with regular languages and the regular expressions that define them and show the behavioral equivalence between regular languages and finite state machines (FSM). The next class of languages are context free grammars and in order to recognize them, we need a more powerful machine in the form of a push-down automata (PDA). The next language is context sensitive languages. We conclude the study of formal languages with recursively enumerable languages which cannot be recognized by an FSM nor a PDA, but require the power of a Turing machine, the machine which defines what IS computable. The Turing machine is formally defined and shown how it can be implemented in hardware.

# Second Half of the Course: Finite State Machine Composition and Decomposition

This course is not concerned with detailed finite state machine (FSM) design as is taught in undergraduate Digital Design, but rather in the behavioral equivalence of the lowest class (but still very powerful) FSMs of different sizes. Once behavioral equivalence of these regular language recognizers is defined, we learn how to combine FSMs using parallel and serial compositions on the way to learning how to decompose larger machines into smaller ones.

The underlying mathematics is a set-theoretic description of a machine as a quintuple of sets. With this mathematical formulation, we can develop several useful things:

- A method for finding the ***single, smallest behavioral equivalent FSM*** to a given machine or proving that it already is the smallest as measured by the number of states by finding and utilizing a reduction homomorphism
- A method for decomposing that single, smallest FSM into a ***provably unique parallel and/or serial decomposition*** by finding and utilizing an image machine isomorphism

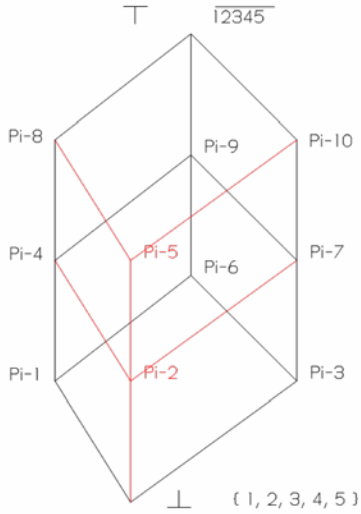


*Figure 1 The provably unique lattice of substitution property partitions from which the machine decomposition of figure 2 was made.*

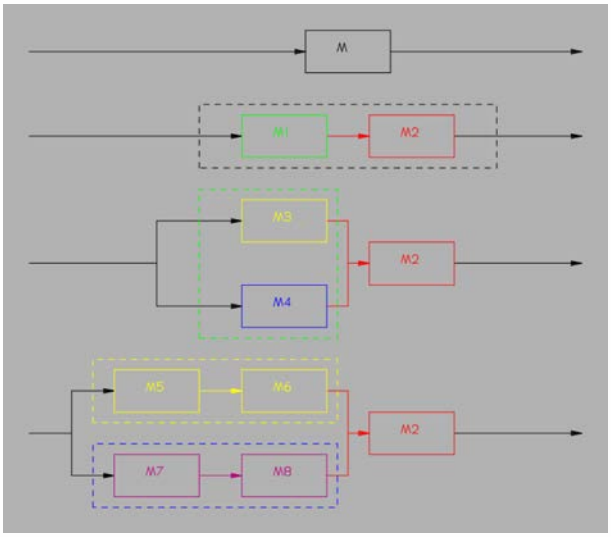


*Figure 2 Block diagram showing the decomposition of a single FSM into a provably unique, maximum parallel/serial decomposition of smaller FSMs.*