Models of Computation

Reading Assignment:
L. Lavagno, A.S. Vincentelli and E. Sentovich,
“Models of computation for Embedded System Design”
Our Design Approach

• Start design process *before* hw-sw partitioning
• Sequence of steps are vital
  – system specification unbiased to implementation
    • describe system behavior at high level
  – Initial functional design
  – verification
  – mapping to target architecture
• Thus, function-architecture codesign is key approach
Proposed design strategy

• Taken from Ref. Of reading assignment.
Design conception to design description

- At functional level, behavior of a system to be implemented is selected and analyzed against a set of specifications
  - Specifications vs. behavior?
    - Specs: I/O relation, set of constraints, system goals
    - behavior: algorithm to realize the function
    - Specs: algorithm itself! (another view)

- Purists view: *Algorithm is the result of implementation decision*
Examples

- Example 1: Let $f(x) = 0$ is a system to be implemented. It is a design decision to use either Newton-Raphson or Gauss-SeidelS relaxation algorithm!
- Example 2: MPEG Encoder design
  Spec: Encoding of compressed stream of data. Any implementation that creates it from the stream is correct. Here the design decision is already there.
Algorithm Design and Analysis

- Algorithm development: Key aspect of system design at functional level
- little work has been done on selection of algorithm based on specifications
- have to have strong correctness properties in critical operations
- Algorithm analysis is more general concept than simulation
- **Important to decide on mathematical model for designer that will support algorithm analysis**
Algorithm Implementation

- Need of intermediate step: transform an algorithm to a set of tractable functional components
- The functional components are to be formally defined to capture the algorithm’s properties
- MoC is key answer to the above!
- Selection of MoC is to be done carefully. (FSM, DF, DES, Comm Seq. Process)
Optimization across MoCs

- Possible to optimize your design across MoC boundaries
  - Encapsulation: interaction between objects in each pair of models who can understand
  - Encompassing Framework: residence of models
  - Orthogonalize concerns: to separate different design aspects such as functions from communication
MoCs
Basic Concepts

• MoC is composed of a description mechanism (syntax) and rules for computation of behavior given the syntax (semantics)
• It is chosen for its suitability: compactness, ability to synthesize, optimize the behavior of implementation
• Most MoCs Permit distributed system of description (a collection of communicating modules), and gives rules of computation of each module (function), and how they communicate.
MoC Primitives

• *Functions*: combination of Boolean functions and synchronous state machines

• *Communications*: queues, buffers, and schedulers
Tagged Signal Model

- A high level abstraction model: Defines processes and their interaction using signals
- Denotational view without any language
- Fundamental entity of TSM: *Event* *(value/tag pair)*
  - *Tags*: temporal behavior
  - A set of *events* is a *signal*

Ex: An event “temperature” may occur any time the sensor interrupts indicating a value (say a range of all possible values in 0-50 degree) at certain intervals.

Do all events have a value to share all the time?
TSM
Tags, Events, Signals

• Given a set of values $V$ and a set of tags $T$, an event is $TXV$
• A signal $s$ is a set of events
• A functional (or deterministic) signal is a (possibly partial) function from $T$ to $V$
• set of all $s = S$, a tuple of $n$ signals $= s$, set of all such tuples $= S^n$
• In timed system, $T$ is totally ordered and in untimed system, $T$ is partially ordered
TSM Processes

• Process P with n signals is a subset of the set of all n-tuples of signals $S^n$
• $s \in S^n$ satisfy the process if $s \in P$
  – *an s that satisfies the process is called behavior of the process.*
• So process is a "set of possible behaviors" or constraints on the set of legal signals.
• Process in a system operate concurrently and constraints imposed are communication or synchronization.
TSM
Processes

• Signals associated with a process may be divided as *input* and *output*
  – process does not determine its input
  – process does determine its output

• Process defines a relation between *input* and *output* signals

• Ex. Say, \( p = \{s_1, s_2, s_3, s_4\} \)
TSM

Process composition

Definition: Process composition in TSM is defined by the intersection of the constraints each process imposes on each signal.

Properties of process preserved by composition:
- functionality (unique output n-tuples for every input n-tuple)
- complete specification (for every input n-tuple, there exists a unique output n-tuple)
Process composition

Q = \{s_1, s_2, s_3, \ldots, s_8\}
TSM
Process composition

- Given a formal model of functional specification and of the properties, three situations may arise:
  - property is *inherent* for model of specification
  - property can be verified *syntactically* for given specification
  - property must be verified *semantically*, for given specification
Functional property examples

• Any design described by Dataflow Network is functional and hence this property need not be checked for this MoC. (Inherent)

• If above design is in FSM, even if the components are functional and completely specified, the result of composition may be either incompletely specified or nonfunctional.
  – This is due to feed-back loop in the composition
  – A syntactical check can find the feed-back paths

• With Petrinets, functionality is difficult to prove.
  – Exhaustive simulation required for checking functionality
Comparing MoC

- System behavior
  - functional behavior, communication behavior, each as of TSM processes
- Process
  - functional behavior and timing behavior
- Function ⇒ how inputs are used for computing output
- Time ⇒ the order in which things happen (assignment of Tags to each event)

*Distinction between Function and Time is not clear in every context as in FSM*
Comparing MoC

• Process function: Boolean
  – control dominated: BDD or DAG
  – data dominated: Data Flow actors, or FFT

• Process State:
  – state is implemented by feedback.
  – behavior ⇒ state transition sequences
Concurrency and communication

• ES has several coordinated *concurrent* processes with *communication* among them

• communication:
  – Explicit: sender forces an order on the events
  – Implicit: sharing of tags, this forces common partial order of events and common notion of state. (common clock)
Basic Time

- ES are usually real-time systems
- Two *events* are *synchronous* if they have the same *tag*.
- Two *signals* are *synchronous* if each *event* in one *signal* is synchronous with an *event* in other *signal* and vice versa
Treatment of Time in Systems

• Discrete-event System: A timed system where the tags in each signal are ordered-isomorphic with natural numbers. (Verilog, VHDL)

• Synchronous System: System in which every signal in the system is synchronous with every other signal in the system

• Discrete-Time System: Synchronous discrete-event system

• Asynchronous System: No two events can have the same tag
  – Asynchronous Interleaved (tags totally ordered)
  – Asynchronous concurrent (partially ordered tags)
Causality

• A casual process has a non-negative time delay from inputs to outputs.
• Strictly casual process has a positive time delay from inputs to outputs.
Communication primitives

- **Unsynchronized**: no coordination, no guarantee of valid read or not overwrite
- **Read-modify-write**: locks data structure during data access (In TSM, events are totally ordered, R-M-W action is one event)
- **Unbounded FIFO buffered**: point-to-point, produced token is consumed only after generated. (TSM context: simple connection where signal is constrained to have totally ordered. If consumer process has unbounded FIFOs on all inputs, all inputs have a total order imposed upon them.
- **Bounded FIFO buffered**: Each input and output signals are internally totally ordered. For buffer size = 1, input and output events must interleaved. For larger size, impose the maximum difference between input or output events occurring in succession.
Models of Computation for reactive systems

• Main MoCs:
  • Finite State Machines (FSM)
  • Data Flow Process Networks
  • Petri Nets
  • Discrete Event
  • Codesign Finite State Machines

• Some main Languages:
  – Esterel, StateCharts, Dataflow Networks