System Design With Synchronous Languages : Esterel

Ellen M. Sentovich
Cadence Berkeley Laboratories

presented by
Roberto Passerone
Outline

- Background and Review
  - system-level specification and design
- Synchrony
- Esterel
  - Background and principles
  - Syntax and semantics
  - Program examples
  - Compilation to hardware and software
- Advanced topics
  - Causality, Optimization, Verification
System Specification and Design

What is a system?

- It produces a steady stream of I/O

  DO_SOMETHING / SOMETHING_DONE;
  DO_MORE / WAIT_A_MINUTE;
  DO_MORE / MORE_DONE;
  I’M_DONE / I’M_NOT;
  DONE_YET? / YES;

- Inconvenient to specify I/O sequences
System Specification: Requirements

- Model
  - Simple and intuitive
  - Accurate enough model of physical reality
  - Mathematically efficient
  - General enough to allow different implementation styles

- Language
  - Accurate expression of the model
  - Convenient and efficient expression of the design

- Analysis

- Implementation
  - compilation to hardware, software
System Specification Issues

- Reactivity
  - reactive, interactive, transformational
- Control versus Data Flow
- Composition
- Temporal behavior
  - sequencing and concurrency
  - preemption
- Communication mechanisms
  - timing (synchronous, asynchronous)
  - data (buffer, queue)
Reactive Systems

- Reactivity
  - continually react at a speed determined by the environment
- Concurrency
  - at least between system and environment
- Strict time and reliability requirements
- Deterministic
- Mixed hardware/software implementations
- Examples:
  - control, supervision, signal-processing, communication protocols, man-machine interfaces
Classical Approaches

- Deterministic automata
  - fast implementation (a single transition is evaluated)
  - verification available
  - too flat: cumbersome and error-prone specification

- Petri-net-based models
  - inherent concurrency
  - lack of hierarchy

- Communicating processes
  - explicit communication and synchronization
  - non-determinism with vague semantics, dependent on implementation
Synchronous Languages

- Invented for the design of reactive kernels
  - not the interactive interface or data management
- Programmer views system as an instantaneous reactor to events
- Program behavior is completely deterministic
- Attempt to resolve concurrency and non-determinism
- Physical time is replaced by multi-form time
  - “stop within 10 seconds”
  - “stop within 100 meters”
Synchronous Instant

- An *instant* is a logical instant
- At each instant, read/compute/write at once
- History of a system: ordered sequence of instants
- Precedence
  - within an instant, actions are performed in order
  - micro-steps
  - synchronous languages “compile away” micro-steps

THUS: communication is instantaneous; computation is internally ordered
Synchronous Hypothesis

- Zero-delay:

  “The reaction of a system to its environment takes zero time.”

  or

  “The reaction of a system takes NEGLIGIBLE time with respect to its environment.”

- **Instantaneous** reaction to events:
  - Read inputs/compute/write outputs happens *instantaneously*
The Zero-Delay Assumption

- Almost nothing takes zero time!
- Assumption must be checked on the final implementation
  - Easily implemented in hardware with proper clocking
  - Easily verified for SW with straight-line code
Esterel Overview

- Basic principles
- Syntax
  - Kernel statements; Programming style
- Execution semantics
- Examples
- Program analysis and verification
Esterel: Background

- Esterel is one of a set of synchronous languages developed in France:
  - Esterel: reactive control
  - Lustre, Signal: data flow
  - Argos, SynchCharts: graphical

- Esterel development goal: natural expression of control
  - Specific statements to deal with time, preemption
  - Departure from concurrency as interleaving and rendez-vous to concurrency as *instantaneous* propagation of control
Esterel: Background

- Synchronous programming environment...
  - Language:
    - For control-dominated, reactive systems
    - Constructs for sequencing, concurrency, preemption
  - Compiler:
    - Produces sorted Boolean equations
    - Causality checking, symbolic debugging, verification
    - Implementation as C-code or digital circuits
Esterel: Basic Principles

- Synchronous hypothesis: instantaneous communication
- Communication via broadcast signals (event)
  - signals, sensors, variables
  - pure, valued
  - Boolean and arithmetic operators
- Modularity: program = collection of modules
- Concise programs: write things once!!
- Concurrency, sequencing, preemption
- Well-defined semantics
Overview of Syntax

- Module definition
  
  module goofy: inputs, outputs, code.

- Signal reading/writing
  
  emit s  emit s(value)  present s then p else q

- Basic control and looping:
  
  halt  loop p end  run module

- Sequencing, concurrency
  
  p ; q  p || q

- Preemption
  
  suspend p when s  abort p when s

  weak abort when s  abort when immediate s
Syntax: await statement

- Most basic signal control statement
  ```plaintext
  await S
  ```

- Equivalent to
  ```plaintext
  abort
  loop
  halt
  end
  when S
  ```

- Note: "Await" always stops
  - consider await S; await S
  - versus await immediate S; await immediate S
Variables and Ordering

- Variable manipulation (local)
  \[ V := \text{value} \quad \text{if} \ V = \text{value} \ \text{then} \ p \ \text{else} \ q \]

- Variable computations : ordered
  
  \[
  \begin{align*}
  V &:= 5; \\
  V &:= V+1; \\
  Y &:= V; \\
  \text{emit} S(V) 
  \end{align*}
  \]

- Signal emissions (communication) : unordered
  
  Legal: emit S; emit S
  Illegal: emit S(3); emit S(5) (non-deterministic)

- Signal computations : ordered
  
  Well-defined: emit S; present S
  Undefined: present S then emit S
The ABRO Example

- “Wait until both A and B have occurred, then output O, unless the reset R occurs”

Number of states is exponential in inputs
The ABRO Example in Esterel

- *Wait until both A and B* $\Rightarrow$ concurrency
- *Unless R* $\Rightarrow$ preemption

```
loop
  abort
  [await A || await B] ;
  emit 0
  when R
end
```

**Write things ONCE !!**

**Code size is linear in inputs**
The ABRO Example in Statecharts
The Seat Belt Example

If the driver turns on the key, and
does not fasten the seat belt within 5 seconds
then an alarm beeps for 5 seconds, or
until the driver fastens the seat belt, or
until the driver turns off the key

input KEY_ON, END_TIMER, KEY_OFF, BELT_ON, RESET;
output START_TIMER, BEEP;
loop
  abort
  present KEY_ON then
    emit START_TIMER(5); await END_TIMER;
    emit START_TIMER(5); abort sustain BEEP when END_TIMER;
  end
  when KEY_OFF or BELT_ON
each RESET
Semantics

- **Denotational semantics**
  - defines a function for each piece of syntax

- **Behavioral semantics**
  - defines the mutations of the program due to the execution of a particular statement given a particular input

- **Operational small-step semantics**
  - like the one above, but at a more refined level

- **Axiomatic semantics**
  - defines the properties that must hold before and after each statement
Behavioral semantics
Compilation of Esterel Programs

- Esterel program \(\Rightarrow\) extended finite state machine
  - finite # inputs, deterministic reaction \(\Rightarrow\) finite program
  - FSM + data computations
- Can be compiled to a single automata - EXPENSIVE!!
  - exhaustive exploration of set of control states
  - Esterel v3
- Can be translated to sorted Boolean equations
  - Esterel v5
- State machine is represented implicitly
Translation to Boolean Circuits

- Structural translation
- Network of interconnected cells
- Implementation of control:
  - boot signal: latch, initially 1, thereafter 0
  - control signal runs through the circuit
- Each cell contains
  - signal input/output; control input/output
  - suspend, kill, resume signals
  - return codes (nested preemption)
- One register per halting statement
Translation to Boolean Circuits

- present s then p else q

- halt
Graphical Symbolic Debugging

- Xes
Causality

- Esterel programs may not be causal
- Program analysis:
  - all present signals must be emitted somewhere
  - a solution is a consistent set of assignments (presence/absence) to the signals
- Non-reactive program:
  
  present X else emit X end

- Non-deterministic program:
  
  present X then emit X end
Logical Correctness

- **Logical correctness**: reactive and deterministic
- This is not enough for causality!
- Logically correct, but difficult:
  ```plaintext
  present X then emit X else emit X
  ```
- A logically correct but strange program (0 + 0 ≠ 0!):
  ```plaintext
  present 01 then
      present 02 then emit 02 end
      else emit 01 end
  ||
  present 02 then
      present 01 then emit 01 end
      else emit 02 end
  ```
Constructive Causality

- Logical correctness
  - sound semantics; not sufficient
  - not intuitive for designers
  - computationally complex

- Operational constructive causality:
  - Step through program analyzing tests, emits, signal statuses
  - derive facts from facts; no future predicting
  - divide signal statuses into must, must not, can
  - permits program interpretation
Executing Constructive Causality

- Example

```
module P1:
  input I;
  output O;
  signal S1, S2 in
    present I then emit S1 end
    ||
    present S1 else emit S2 end
    ||
    present S2 then emit O end
  end signal
end module
```

- Instantaneous dialog
Electrical Constructive Causality

- Precisely equivalent to electrically stable, combinationally acyclic circuits
  
  \[
  \text{present } x \text{ then emit } x \text{ end}
  \]
  
  \[
  \text{present } x \text{ else emit } x \text{ end}
  \]
Causality Analysis

- Analysis carried out on the sequential circuit equivalent
- Combinational loops are broken; three-valued simulation performed
- Reachable state computation; BDD representation
- Check that all feedback wires are well-defined
- Expensive! Interpretation option in Esterel
Causality: A Real Example

- Bus arbitration

![Diagram of bus arbitration](image)
Optimization

- Standard logic synthesis techniques can be applied to generated hardware
- Finite state machine optimization and analysis
- Esterel state encoding:
  - locally group-hot
  - efficient implementation
  - far more latches than the minimum
- Latch minimization algorithms
  - exploit the latch/logic tradeoff
  - progressively more powerful algorithms
Latch Optimization

- Replacement of latches with logic
  - compute the reachable state set
  - compute sets of latches that can be replaced
    - single latch removal: \( R_i \ast R_i' = 0 \)
    - 2-for-1 removal: \( R_{ij} \ast R_{ij'} + R_{ij'} \ast R_{ij} = 0 \)

Reachable states
Karnaugh map
Latch Optimization

- Generate decoding logic
  - single-latch: remove latch
  - 2-for-1: remove two latches, add logic for one

- Generate encoding logic
  - single-latch: \( E = R \) (dependent on other latches)
Latch Optimization : Exclusive Sets

- Mutually exclusive latch activity
  
  ```
  await s1 || await s2 || await s3
  
  ;
  
  await s4 || await s5 || await s6
  ```

- Latch corresponds to each wait

- \((s1, s2, s3) \neq (s4, s5, s6)\)

- Replace 6 latches with 3 plus a multiplexing latch
Latch Optimization: Results

- Esterel encoding maintains the structure of the initial program
- Single latch removal is always beneficial
- More sophisticated algorithms exploit latch/logic tradeoff
- Cost functions depend on circuit size, BDD size
- Typically 2-for-1 is performed until logic becomes prohibitively expensive
- Verification times are improved
Verification

- Simulation and debugging
- Synchronous observers
  - specify an Esterel module to monitor the system and report errors
  - specify an Esterel module to model the environment
  - perform symbolic reachability analysis
- Temporal logic formulas translated to Esterel modules
  - TempEst work at U Texas
Applications

- Avionics, automotive, manufacturing
  - Dassault: landing gear control; all software
  - Renault

- Communication protocols, glue logic
  - A T & T: switching software; safety properties
  - Motorola for bus interface circuits
  - Daimler-Benz: communication protocol for safe in-vehicle communication

- Man-machine interfaces

- Hardware/software codesign