Tutorial: a Practical Introduction to using Event-B for Complex Hardware and Embedded System Specification and Design

John Colley
FDL 2012
Vienna
Introduction

• Background to Event-B
• Event-B in the Design/Verification Flow
• Complex hardware specification/verification
  – Pipelines
  – Elastic Buffering
• Embedded system specification/verification
  – Temporal Modeling in Cyber-physical systems
  – Animating and Model Checking Event-B models
• Assertion-based verification
  – Deriving assertions from the specification
• Summary
Background to Event-B

• **Event-B** is a formal method for system-level modelling and analysis which uses
  – set theory as a modelling notation
  – refinement to represent systems at different abstraction levels
  – mathematical proof to verify consistency between refinement levels.

• The **Rodin Platform** is an Eclipse-based IDE for Event-B
  – provides support for refinement and mathematical proof
  – open source

• **ProB** is an animator and model checker in the Rodin environment
Industrial Deployment of Event-B

• **Deploy** (FP7 – completed 2012)
  – **Bosch** have been working on developing a cruise control system and a start-stop system
  – **Siemens Transportation** have been working on train control and signalling systems
  – **Space Systems Finland** have been working on part of the BepiColombo space probe and on Attitude and Orbit Control System software (AOCS)
  – **SAP** have been working on analysis of business choreography models
  – **Systerel** are working on railway and aerospace systems

• **ADVANCE** (FP7 – started Oct 2011)
  – Event-B for Cyber-Physical Systems
Event-B in a Design /Verification Flow

- Requirements/Safety Analysis
- Specification Refinement
- Architectural Refinement
- Implementation Verification
Event-B in a Design /Verification Flow

Requirements/Safety Analysis

Specification Refinement

Architectural Refinement

Implementation Verification

Event-B Refinement
Event-B in a Design /Verification Flow

- Requirements/Safety Analysis
- Specification Refinement
- Architectural Refinement
- Implementation Verification

Event-B Refinement

Code and Assertion Generation
A MicroProcessor Pipeline

• Each pipeline stage is a process running concurrently with all the other stages
• Communication is by shared variables (pipeline registers)
• New high-level languages speed up design
  – Bluespec
    • Guarded atomic actions
    • High-level synthesis to RTL
• But verification is still
  – performed on low-level RTL description
  – predominantly test-based
Generic Operations
Load
Store
Branch
ArithRR
ArithImm

Pipeline Stages
Instruction Fetch (IF)
Instruction Decode (ID)
Execute (EX)
Memory Access (MEM)
Writeback (WB)

- pipeline register
Pipeline Verification Goals

• Start Verification at the Specification Level
• Explore micro-architectural alternatives at the specification level
• Close the gap between specification and implementation
• Exploit synergy with Bluespec
• Incorporate proof-based techniques into the established SoC verification flow
Specifying an Arithmetic Instruction

class context PIPEC

constants Register Rr Ra Rb ArithRROp

sets Op // Operations

axioms
@axm1 Register ⊆ N // Processor Register Identifier
@axm2 Rr ∈ Op → Register // Destination Register
@axm3 Ra ∈ Op → Register // First Source Register
@axm4 Rb ∈ Op → Register // Second Source Register
@axm5 ArithRROp ⊆ Op // Register/Register Arithmetic Ops
end
The Abstract Machine

machine PIPEM sees PIPEC

variables Regs

invariants
  @inv1 Regs ∈ Register → ℤ // The Processor Register File

events
  event INITIALISATION
    then
      @act1 Regs = Register × {0}
  end

  event ArithRR
    any pop
    where
      @grd1 pop ∈ ArithRROp
    then
      @act1 Regs(Rr(pop)) = Regs(Ra(pop)) + Regs(Rb(pop))
  end
end
The Abstract Architecture

pop

Regs

ArithRR
event ArithRR
  any pop
  where
    @grd1 pop ∈ ArithRROp
  then
    @act1 Regs(Rr(pop)) = Regs(Ra(pop)) + Regs(Rb(pop))
  end
First Refinement: a two-stage pipeline

**Diagram:**
- IFIDEX
- EXWB pipeline registers
- WB

**Process:**
1. **ppop**
2. IFIDEX → IFIDEX → EXop → EXALUout → WB
3. IFIDEX → EXop → EXALUout → WB

**Stage:**
1. **First Refinement:** a two-stage pipeline
Pipeline Feedback and Interleaving

PPop

\[ \text{v2} \rightarrow \text{e1i} \rightarrow \text{v1} \rightarrow \text{e2j} \]

\text{e2j followed by e1i (e2j;e1i) is equivalent to e1i || e2j}

PPop

\[ \text{v2} \rightarrow \text{e1i} \rightarrow \text{v1} \rightarrow \text{e2j} \]

\text{there is NO interleaving that represents e1i || e2j}
event EXWBnoRAW refines ArithRR
  any ppop
  where
  @grd1 EXop ∈ ArithRROp
  @grd2 ppop ∈ ArithRROp
  with
  @pop pop = EXop
  then
  @act1 Regs(Rr(EXop)) = EXALUoutput
  @act2 EXALUoutput = Regs(Ra(ppop)) + Regs(Rb(ppop))
  @act3 EXop = ppop
end
event ArithRR any pop where @grd1 pop ∈ ArithRROp
then
@act1 Regs(Rr(pop)) = Regs(Ra(pop)) + Regs(Rb(pop))
end

event EXWBnoRAW refines ArithRR any ppop where @grd1 EXop ∈ ArithRROp @grd2 ppop ∈ ArithRROp with @pop pop = EXop
then
@act1 Regs(Rr(EXop)) = EXALUoutput
@act2 EXALUoutput = Regs(Ra(ppop)) + Regs(Rb(ppop))
@act3 EXop = ppop
end
The Gluing Invariant

\[ @\text{inv3} \quad \text{EXALUoutput} = \text{Regs}(Ra(\text{EXop})) + \text{Regs}(Rb(\text{EXop})) \]
Parallel Execution must detect the potential Read After Write (RAW) Hazard

The Gluing Invariant

@inv3 \text{EXALUoutput} = \text{Regs}(\text{Ra}(\text{EXop})) + \text{Regs}(\text{Rb}(\text{EXop}))
event EXWBnoRAW refines ArithRR
any ppop
where
  @grd1 EXop ∈ ArithRROp
  @grd2 ppop ∈ ArithRROp
  @grd3 Rr(EXop) ≠ Ra(ppop)
  @grd4 Rr(EXop) ≠ Rb(ppop)
with
  @pop pop = EXop
then
  @act1 Regs(Rr(EXop)) = EXALUoutput
  @act2 EXALUoutput = Regs(Ra(ppop)) + Regs(Rb(ppop))
  @act3 EXop = ppop
end
ppop

IFIDEX

EXWB pipeline registers

EXop

EXALUout

WB

Reg

Reg

forwarding

Reg

Reg

Reg
event EXWBaRAW refines ArithRR
  any ppop
  where
    @grd1 EXop ∈ ArithRROp
    @grd2 ppop ∈ ArithRROp
    @grd3 Rr(EXop) = Ra(ppop)
    @grd4 Rr(EXop) ≠ Rb(ppop)
  with
    @pop pop = EXop
  then
    @act1 Regs(Rr(EXop)) = EXALUoutput
    @act2 EXALUoutput = EXALUoutput + Regs(Rb(ppop))
    @act3 EXop = ppop
Event WB where EXop ∈ ArithRROp then Regs(Rr(EXop)) = EXALUoutput end

event EXWBaRAW refine any ppop where
  @grd1 EXop ∈ ArithRROp
  @grd2 ppop ∈ ArithRROp
  @grd3 Rr(EXop) = Ra(ppop)
  @grd4 Rr(EXop) ≠ Rb(ppop)
with
  @pop pop = EXop
then
  @act1 Regs(Rr(EXop)) = EXALUoutput
  @act2 EXALUoutput = EXALUoutput + Regs(Rb(ppop))
  @act3 EXop = ppop
end
Forwarding and Centralised Stalling

• As the pipeline gets longer
  – More forwarding tracks are required
    • the feedback tracks get longer
  – Centralised stalling to manage branching becomes more complex and difficult to verify
    • the feedback tracks get longer

• Synchronous Elastic Buffers provide an alternative solution
  – Latency insensitive
  – Distributed stalling
  – First used by Intel to meet timing requirements
The SELF Protocol
Connecting two Elastic Components

Value DATA

stop
valid
elastic buffer

Value DATA
Recall Abstract Machine Micro-architecture

\[
\text{event } \text{ArithRR} \\
\text{any } \text{pop} \\
\text{where} \\
\quad @\text{grd1} \text{ pop } \in \text{ArithRROp} \\
\text{then} \\
\quad @\text{act1} \text{ Regs}(\text{Rr(pop)}) = \text{Reg}(\text{Ra(pop)}) + \text{Reg}(\text{Rb(pop)}) \\
\text{end}
\]
Refinement with Synchronous Elastic Buffers
Abstract Synchronous Elastic Buffer
Refinement with Synchronous Elastic Buffers
event EXWBnoRAW refines ArithRR

any pppop

where

@grd1 pppop ∈ ArithRROp
@grd2 obuf1(ord1) ∈ ArithRROp
@grd3 obuf2(ord2) ∈ ArithRROp
@grd4 Valid1 = TRUE
@grd5 Rr(obuf1(ord1)) ≠ Ra(pppop)
@grd6 Rr(obuf1(ord1)) ≠ Rb(pppop)
@grd7 Rr(obuf2(ord2)) ≠ Ra(pppop)
@grd8 Rr(obuf2(ord2)) ≠ Rb(pppop)
@grd9 Valid2 = TRUE

with

@pop pop = obuf1(ord1)

then

@act1 Regs(Rr(obuf1(ord1))) = vbuf1(vrd1)
@act2 obuf1(owr1) = obuf2(ord2)
@act3 vbuf1(vwr1) = vbuf2(vrd2)
@act4 vbuf2(vwr2) = Regs(Ra(pppop)) + Regs(Rb(pppop))
@act5 obuf2(owr2) = pppop

... // update buffer indices

end
event EXWBnoRAW refines ArithRR

any ppop

where

@grd1 ppop ∈ ArithRROp
@grd2 obuf1(ord1) ∈ ArithRROp
@grd3 obuf2(ord2) ∈ ArithRROp
@grd4 Valid1 = TRUE
@grd5 Rr(obuf1(ord1)) ≠ Ra(pppop)

@act1 Regs(Rr(obuf1(ord1))) := vbuf1(vrd1)
@act2 obuf1(owr1) := obuf2(ord2)
@act3 vbuf1(vwr1) := vbuf2(vrd2)
@act4 vbuf2(vwr2) := Regs(Ra(pppop)) + Regs(Rb(pppop))

... // update buffer indices
event EXWBRAWa refines ArithRR

any $pppop$

where

@grd1 $pppop \in \text{ArithRROp}$
@grd2 $\text{obuf1}(\text{ord1}) \in \text{ArithRROp}$
@grd3 $\text{obuf2}(\text{ord2}) \in \text{ArithRROp}$
@grd4 $\text{Valid1} = \text{TRUE}$
@grd5 $\text{Rr}(\text{obuf1}(\text{ord1})) = \text{Ra}(pppop)$
@grd6 $\text{Rr}(\text{obuf1}(\text{ord1})) \neq \text{Rb}(pppop)$
@grd7 $\text{Rr}(\text{obuf2}(\text{ord2})) \neq \text{Ra}(pppop)$
@grd8 $\text{Rr}(\text{obuf2}(\text{ord2})) \neq \text{Rb}(pppop)$
@grd9 $\text{Valid2} = \text{TRUE}$

with

@pop $pop = \text{obuf1}(\text{ord1})$

then

@act1 $\text{Regs}(\text{Rr}(\text{obuf1}(\text{ord1}))) = \text{vbuf1}(\text{vrd1})$
@act2 $\text{obuf2}(\text{owr2}) = pppop$
@act3 $\text{obuf1}(\text{owr1}) = \text{obuf2}(\text{ord2})$
@act4 $\text{vbuf1}(\text{vwr1}) = \text{vbuf2}(\text{vrd2})$
@act5 $\text{Valid2} = \text{FALSE}$

... // update buffer indices

end
event EXstallWB refines ArithRR

any pppop

where

@grd1 pppop ∈ ArithRROp
@grd2 obuf1(ord1) ∈ ArithRROp
@grd3 obuf2(ord2) ∈ ArithRROp
@grd4 Valid1 = TRUE
@grd5 Rr(obuf1(ord1)) ≠ Ra(pppop)
@grd6 Rr(obuf1(ord1)) ≠ Rb(pppop)
@grd7 Rr(obuf2(ord2)) ≠ Ra(pppop)
@grd8 Rr(obuf2(ord2)) ≠ Rb(pppop)
@grd9 Valid2 = FALSE

with

@pop pop = obuf1(ord1)

then

@act1 Regs(Rr(obuf1(ord1))) = vbuf1(vrd1)
@act2 obuf1(owr1) = obuf2(ord2)
@act3 vbuf1(vwr1) = vbuf2(vrd2)
@act4 vbuf2(vwr2) = Regs(Ra(pppop)) + Regs(Rb(pppop))
@act5 obuf2(owr2) = pppop
@act6 Valid1 = FALSE
@act7 Valid2 = TRUE

... // update buffer indices

end
event EXWBstall

any pppop

where

@grd1  pppop ∈ ArithRROp
@grd2  obuf2(ord2) ∈ ArithRROp
@grd2  obuf2(ord2) = obuf1(ord1)
@grd3  Valid1 = FALSE
@grd4  Rr(buf1(ord1)) ≠ Ra(pppop)
@grd5  Rr(buf1(ord1)) ≠ Rb(pppop)
@grd6  Rr(buf2(ord2)) ≠ Ra(pppop)
@grd7  Rr(buf2(ord2)) ≠ Rb(pppop)
@grd8  Valid2 = TRUE

then

@act1  vbuf2(vwr2) = Regs(Ra(pppop)) + Regs(Rb(pppop))
@act2  obuf2(owr2) = pppop
@act3  vbuf1(vwr1) = vbuf2(vrd2)
@act4  obuf1(owr1) = obuf2(ord2)
@act5  Valid1 = TRUE

... // update buffer indices

end
event EXWBstall
any ppop
where
  @grd1 ppop ∈ ArithRROp
  @grd2 obuf2(ord2) ∈ ArithRROp
  @grd2 obuf2(ord2) = obuf1(ord1)
  @grd3 Valid1 = FALSE
  @grd4 Rr(obuf1(ord1)) ≠ Ra(ppop)
  @grd5 Rr(obuf1(ord1)) ≠ Rb(ppop)
  @grd6 Rr(obuf2(ord2)) ≠ Ra(ppop)
  @grd7 Rr(obuf2(ord2)) ≠ Rb(ppop)
  @grd8 Valid2 = TRUE
then
  @act1 vbuf2(vwr2) := Regs(Ra(ppop)) + Regs(Rb(ppop))
  @act2 obuf2(owr2) := ppop
  @act3 vbuf1(vwr1) := vbuf2(vrd2)
  @act4 obuf1(owr1) := obuf2(ord2)
  @act5 Valid1 := TRUE
... // update buffer indices
end
Shared Event Pipeline Decomposition
Second Refinement: MACHINE 1

MACHINE 1

MACHINE 2

MACHINE 3

ppop

valid

stop
Pipelining Summary

• Micro-architectural exploration is raised to the Specification Level using Event-B
• An alternative to forwarding and centralised stalling has been explored using Synchronous Elastic Buffers
• Latency Insensitivity is introduced at Low Cost
• Track lengths are reduced
• Synchronous Elastic Buffers allow performance goals to be met in a verifiable way
• Verification is raised to the Specification Level
Temporal Modeling in Cyber-physical systems

• Simulating Formal Models
• Modeling Timing Cycles
  – Component Modes
  – Generalised Update/Evaluation Modes
• A Simple Example
• Summary
Temporal Modeling in Cyber-physical Systems: Requirements

- Distributed Function and Control
- Managing Safety Hazards
- Verifying the relationships between Inputs and Outputs
Distributed Function and Control

Controller 1

Controller 2

B

D

C1

C2

C3
Distributed Function and Control

Must Model the Communication and Synchronisation of the Concurrent Processes
Managing Safety Hazards

• Plant model Evaluates
• Potential Hazards are Detected
• The Controller manages the Hazards
• The Controller predicts the future behaviour of the Plant
• Loop must be generalised for multiple controllers and distributed Plant
Verifying the relationships between Inputs and Outputs

• DO-178C Formal Supplement
  – Must show that
    1. Outputs fully satisfy Inputs
    2. Each Output data item is necessary to satisfy some Input data item (*No unintended behaviour*)
  – Must show that
    • Input/Output specification is preserved by chosen implementation architecture

• ACSL
  – ANSI ISO C Specification Language
  – Code annotations – used by Airbus
Simulating Formal Models

• Abstract Model(s) may be *untimed*
• Refined Models represent Concurrent, Communicating Processes
  – will need to introduce some notion of a **tick**
    • Cycle-based execution
    • Timed execution of *Delays* and *Deadlines*
• ProB has the notion of the *next state*
  – an event is executed (LTL $X$)
• We need the notion of the *next tick*
Modeling Timing Cycles: Component Modes

• Eg for Hazard Analysis
  – *Plant* Mode
  – Detect Mode
  – *Controller* Mode
  – Predict Mode

• Necessary to define an *ordering* on the modes

• The Plant may need to evaluate at a much higher rate than the Controller
Modeling Timing Cycles: Update/Evaluate Modes

• Used by many Commercial Discrete Event Simulators
  – SystemC
  – Verilog/VHDL

• Supports arbitrary topology complexity

• No zero-delay communication between components
  – Components can evaluate in any order

• Components “suspend” between wake-ups
  – Input change
  – Self wake
  – Components can evaluate at different rates

• Discrete Time, Cycle-based or both
Discrete Event Simulation

COMPONENT VIEW

Components: A, B, C, D (processes)
Connections: C1, C2 (unidirectional)
Ports: IN  OUT

SIMULATOR API

GetValue(port)
HasChanged(port)
SetValue(OUT port, val, delay)
ScheduleEval(component, delay)
The Two-list Simulation Algorithm

\[ t = 0 \]

update list

\[ t = n \]

evaluation list

A \rightarrow C1 \rightarrow B

C \rightarrow C2 \rightarrow D
Simulating Models without Discrete Delays – *Unit Delay*

Each evaluate/update cycle advances time by one *tick*
A Simple Arithmetic Example

The Abstract Specification – *Output* as a function of *Inputs*

```
constants Inputs In1 In2

sets Parameters

axioms
  @axm1 Inputs ⊆ Parameters
  @axm2 In1 ∈ Inputs → ℕ
  @axm3 In2 ∈ Inputs → ℕ
end

event AddInc
  any p
  where
    @grd1 p ∈ Inputs
    @grd2 In1(p) ∈ ℕ
    @grd3 In2(p) ∈ ℕ
  then
    @act1 v := In1(p) + In2(p) + 1
end
```
event AddInc
  any $p$
  where
  @grd1 $p \in \text{Inputs}$
  @grd2 $\text{In1}(p) \in \mathbb{N}$
  @grd3 $\text{In2}(p) \in \mathbb{N}$
  then
  @act1 $v = \text{In1}(p) + \text{In2}(p) + 1$
end

Channel: $\text{sum}$
Delay = 2
The Implementation Architecture

event AddInc
    any p
    where
    @grd1 p ∈ Inputs
    @grd2 In1(p) ∈ ℕ
    @grd3 In2(p) ∈ ℕ

Does the chosen Architecture Implement the Abstract Specification?

*Output* as a function of *Inputs* (DO-178C)
Modelling Channels with Delay in Event-B

• A Channel is a Set of Schedules
• A Schedule comprises
  – a Delay (greater than or equal to 0)
  – a Value (optional)
  – the Input Values that correspond to the Output Value (optional)
Writing to a Channel

• A Channel write is accomplished by creating a new schedule with a delay of at least one
• Multiple Schedules may be added
  – Prevent multiple schedules for same time \textit{OR}
  – Choose one non-deterministically

\{2, 1, 6\}
The Update/Evaluate Cycle

• **Update** is modeled using a single Event-B event
• **Evaluate** is represented by one or more enabled Component Events
Evaluation Mode

- All Components, where one or more of their *Input* Channels has a schedule with delay 0, *resume* (at least one Component event is enabled and the Update event is disabled)
  - Change local state
  - Create new Schedules on Output Channels
  - *Suspend*
Update Mode

• The Update Event is enabled when all the Components have been evaluated
  – Schedules with 0 delay are deleted
  – All other schedule delays are decremented
  – The current tick is therefore complete

• The Update Event is re-enabled if no schedule has 0 delay, resulting in another tick
Our Example: The First Refinement

Modelling the Channel

@inv3 sum_value ∈ Schedule ↦ ℕ
@inv4 sum_delay ∈ Schedule ↦ ℕ
@inv5 sum_inputs ∈ Schedule ↦ Inputs

Channel: \textit{sum}

Add

Increment

Delay = 2
The Add and Increment Components

event Add
  any p s
  where
    @grd1 p ∈ \textit{Inputs}
    @grd2 \textit{ln1}(p) ∈ \mathbb{N}
    @grd3 \textit{ln2}(p) ∈ \mathbb{N}
    @grd4 s ∉ \text{dom}(\textit{sum\_delay})
    @grd5 \text{adder\_evaluated} = \text{FALSE}
  then
    @act1 \text{sum\_value}(s) = \textit{ln1}(p) + \textit{ln2}(p)
    @act2 \text{sum\_delay}(s) = 2
    @act3 \text{sum\_inputs}(s) = p
    @act4 \text{adder\_evaluated} = \text{TRUE}
end

event Increment refines AddInc
  any s
  where
    @grd1 s ∈ \text{dom}(\textit{sum\_delay})
    @grd2 \textit{sum\_delay}(s) = 0
    @grd3 \text{injector\_evaluated} = \text{FALSE}
  with
    @p p = \text{sum\_inputs}(s)
  then
    @act1 v = \text{sum\_value}(s) + 1
    @act2 \text{inc\_sum} = \text{sum\_value}(s)
    @act3 \text{injector\_evaluated} = \text{TRUE}
end

@inv3 sum\_value ∈ \textit{Schedule} ↦ \mathbb{N}
@inv4 sum\_delay ∈ \textit{Schedule} ↦ \mathbb{N}
@inv5 sum\_inputs ∈ \textit{Schedule} ↦ \textit{Inputs}
The Update Event/Synchronisation

event Update
  where
  @grd1 adder_evaluated = TRUE
  @grd2 0 ∉ ran(sum_delay) ∨ incrementer_evaluated = TRUE
  then
  @act1 adder_evaluated = FALSE
  @act2 incrementer_evaluated = FALSE
  @act3 sum_delay = λi·i ∈ dom(sum_delay) ∧ sum_delay(i) > 0 | sum_delay(i) − 1
  @act4 sum_value = λi·i ∈ dom(sum_value) ∧ i ∈ dom(sum_delay) ∧ sum_delay(i) > 0 | sum_value(i)
  @act5 sum_inputs = λi·i ∈ dom(sum_inputs) ∧ i ∈ dom(sum_delay) ∧ sum_delay(i) > 0 | sum_inputs(i)
  end

event Add
  any p s
  where
  @grd1 p ∈ Inputs
  @grd2 ln1(p) ∈ ℕ
  @grd3 ln2(p) ∈ ℕ
  @grd4 s ∉ dom(sum_delay)
  @grd5 adder_evaluated = FALSE
  then
  @act1 sum_value(s) = ln1(p) + ln2(p)
  @act2 sum_delay(s) = 2
  @act3 sum_inputs(s) = p
  @act4 adder_evaluated = TRUE
  end

event Increment refines AddInc
  any s
  where
  any s
  where
  @grd1 s ∈ dom(sum_delay)
  @grd2 sum_delay(s) = 0
  @grd3 incrementer_evaluated = FALSE
  with
  @p p = sum_inputs(s)
  then
  @act1 v = sum_value(s) + 1
  @act2 inc_sum = sum_value(s)
  @act3 incrementer_evaluated = TRUE
  end
The Gluing Invariant

Represented as the function of Inputs to Output as preserved in the Schedules

@inv9 ∀ s·s ∈ dom(sum_value) ⇒ sum_value(s) = ln1(sum_inputs(s)) + ln2(sum_inputs(s))
The Second Refinement – Remove Inputs

**event Update**

*where*

\(@\text{grd1 adder}_\text{evaluated} = \text{TRUE}\)
\(@\text{grd2} 0 \notin \text{ran}(\text{sum}_\text{delay}) \lor \text{incrementer}_\text{evaluated} = \text{TRUE}\)*

*then*

\(@\text{act1 adder}_\text{evaluated} \leftarrow \text{FALSE}\)
\(@\text{act2 incrementer}_\text{evaluated} \leftarrow \text{FALSE}\)
\(@\text{act3 sum}_\text{delay} = \lambda i \cdot i \in \text{dom}(\text{sum}_\text{delay}) \land \text{sum}_\text{delay}(i) > 0 \lor \text{sum}_\text{delay}(i) - 1\)
\(@\text{act4 sum}_\text{value} = \lambda i \cdot i \in \text{dom}(\text{sum}_\text{value}) \land i \in \text{dom}(\text{sum}_\text{delay}) \land \text{sum}_\text{delay}(i) > 0 \lor \text{sum}_\text{value}(i)\)

*end*

**event Add**

*any p s*

*where*

\(@\text{grd1} p \in \text{Inputs}\)
\(@\text{grd2} \text{ln1}(p) \in \mathbb{N}\)
\(@\text{grd3} \text{ln2}(p) \in \mathbb{N}\)
\(@\text{grd4} s \notin \text{dom}(\text{sum}_\text{delay})\)
\(@\text{grd5 adder}_\text{evaluated} = \text{FALSE}\)*

*then*

\(@\text{act1 sum}_\text{value}(s) = \text{ln1}(p) + \text{ln2}(p)\)
\(@\text{act2 sum}_\text{delay}(s) = 2\)
\(@\text{act4 adder}_\text{evaluated} = \text{TRUE}\)

*end*

**event Increment refines AddInc**

*any s*

*where*

\(@\text{grd1} s \in \text{dom}(\text{sum}_\text{delay})\)
\(@\text{grd2} \text{sum}_\text{delay}(s) = 0\)
\(@\text{grd3 incrementer}_\text{evaluated} = \text{FALSE}\)*

*then*

\(@\text{act1} v = \text{sum}_\text{value}(s) + 1\)
\(@\text{act2} \text{inc}_\text{sum} = \text{sum}_\text{value}(s)\)
\(@\text{act3 incrementer}_\text{evaluated} = \text{TRUE}\)

*end*
Cycle-Based Simulation- Represent Schedules as Pairs of Variables

event Update refines Update
  where
  \[ \text{@grd1 } \text{adder\_evaluated} = \text{TRUE} \]
  \[ \text{@grd2 } \text{msg\_rcvd\_on\_sum} = \text{FALSE} \lor \text{incrementer\_evaluated} = \text{TRUE} \]
  then
  \[ \text{@act1 } \text{msg\_rcvd\_on\_sum} = \text{msg\_sent\_on\_sum} \]
  \[ \text{@act2 } \text{sum} = \text{sum\_prime} \]
  \[ \text{@act3 } \text{msg\_sent\_on\_sum} = \text{FALSE} \]
  \[ \text{@act4 } \text{adder\_evaluated} = \text{FALSE} \]
  \[ \text{@act5 } \text{incrementer\_evaluated} = \text{FALSE} \]
  end

event Add refines Add
  any \( p \)
  where
  \[ \text{@grd1 } p \in \text{Inputs} \]
  \[ \text{@grd2 } \text{In1}(p) \in \mathbb{N} \]
  \[ \text{@grd3 } \text{In2}(p) \in \mathbb{N} \]
  \[ \text{@grd4 } \text{adder\_evaluated} = \text{FALSE} \]
  then
  \[ \text{@act1 } \text{sum\_prime} = \text{In1}(p) + \text{In2}(p) \]
  \[ \text{@act2 } \text{msg\_sent\_on\_sum} = \text{TRUE} \]
  \[ \text{@act3 } \text{adder\_evaluated} = \text{TRUE} \]
  end

event Increment refines Increment
  where
  \[ \text{@grd1 } \text{msg\_rcvd\_on\_sum} = \text{TRUE} \]
  \[ \text{@grd2 } \text{incrementer\_evaluated} = \text{FALSE} \]
  then
  \[ \text{@act1 } v = \text{sum} + 1 \]
  \[ \text{@act2 } \text{incrementer\_evaluated} = \text{TRUE} \]
  end
Cycle-Based Simulation- A *Pair* of Gluing Invariants

@inv9 msg_rcvd_on_sum = TRUE ⇒ sum = \text{In1}(\text{sum}\_\text{inputs}) + \text{In2}(\text{sum}\_\text{inputs})

@inv10 msg_sent_on_sum = TRUE ⇒ \text{sum}\_\text{prime} = \text{In1}(\text{sum}\_\text{inputs}\_\text{prime}) + \text{In2}(\text{sum}\_\text{inputs}\_\text{prime})
Animating and Model Checking Event-B models with ProB
event AddInc
any p
where
@grd1 p ∈ Inputs
@grd2 In1(p) ∈ ℕ
@grd3 In2(p) ∈ ℕ
then
@act1 v := In1(p) + In2(p) + 1
end
Cyber-physical Modeling Summary

- Update/Evaluate Simulation semantics can be formalised in Event-B
- Event-B component models can be simulated/co-simulated with third party simulators
- Event-B refinement supports naturally the DO-178C requirement to verify the relationship between Inputs and Outputs at Specification and Implementation Level
- Update/Evaluate modes provides a suitable basis for a Formal Safety Analysis
Assertion-based Verification

• Identify Assertions at the Specification Level
  – Event-B Invariants
    • Abstract Level
    • Concrete Level

• Translate Invariants to
  – PSL
  – SVA

• Translate Synthesised Assertions to Event-B
  – Formal Proof
  – Model Checking

• Assertion Coverage
  – Trace back to Requirements
Summary

• Background to Event-B
• Event-B in the Design/Verification Flow
• Complex hardware specification/verification
  – Pipelines
  – Elastic Buffering
• Embedded system specification/verification
  – Temporal Modeling in Cyber-physical systems
  – Animating and Model Checking Event-B models
• Assertion-based verification
  – Deriving assertions from the specification