Roll Your Own Test Bed for Embedded Real-Time Protocols: A Haskell Experience

Lee Pike, Galois, Inc.
leepike@galois.com

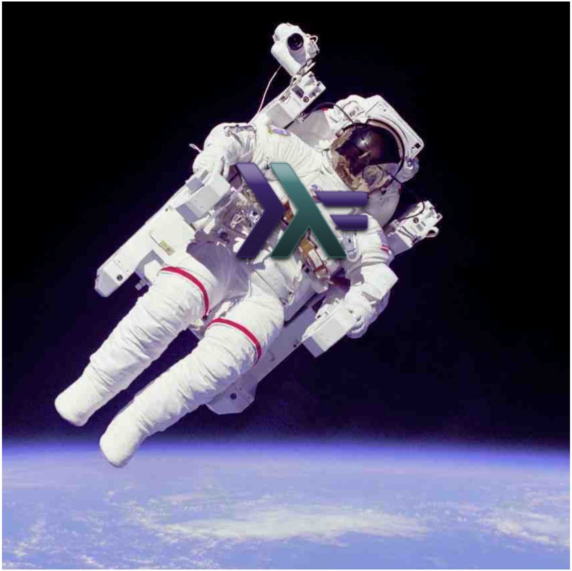
Geoffrey Brown, Indiana University
Alwyn Goodloe, National Institute of Aerospace

September 3, 2009

galois

This is a story about using Haskell...

. . .Haskell in Space!



|galois|

Ok, just kidding.

Goals:

1. Teach you enough about physical-layer protocols to make you dangerous.
2. Tell you how I easily modeled real-time distributed systems in a lazy, pure language.
3. Tell you how I used QuickCheck as a “probability calculator”.

(1) Physical-Layer Protocols

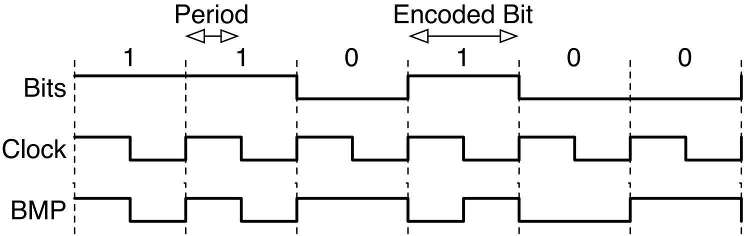
Do you wonder how

- The credit-card slider reads your magnetic stripe?
- The CD player reads your Rolling Stones CD?
- The internet gets transmitted to your computer?

All are achieved using physical-layer protocols: a transmitter `tx` sends a receiver `rx` a bit-stream in real-time.

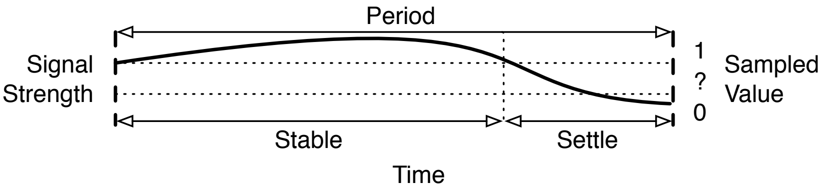
|galois|

Biphase Mark Protocol



|galois|

Signal Strength Over Time



Parameters

```
-- | Realtime input parameters.
data Params = Params
    { tPeriod  :: Time -- ^ Tx's clock period.
    , tSettle  :: Time -- ^ Nominal signal settling time.
    , rScanMin :: Time -- ^ Rx's min scan duration.
    , rScanMax :: Time -- ^ Rx's max scan duration.
    , rSampMin :: Time -- ^ Rx's min sampling duration.
    , rSampMax :: Time -- ^ Rx's max sampling duration.
    } deriving (Show, Eq)
```

Some Constraints

```
paramsConst :: Params -> Bool
paramsConst p =
     0 < tPeriod p -- tPeriod
  && 0 <= tSettle p -- tSettle
  && tSettle p < tPeriod p -- tSettle
  && 0 < rScanMin p -- rScanMin
  && rScanMin p <= rScanMax p -- rScanMax
  && rScanMax p < tStable  -- rScanMax
  && tPeriod p + tSettle p < rSampMin p -- rSampMin
  && rSampMin p <= rSampMax p -- rSampMax
  && rSampMax p < tPeriod p + tStable - rScanMax p
   where tStable = tPeriod p - tSettle p
```

(2) Modeling Real-Time in Haskell

Modeling Real-Time

In a Lazy Functional Language

- **Question:** How do I model distributed real-time behavior in a functional language?

  More precisely, we want to model a partially-synchronous real-time system with possibly non-deterministic bounds on asynchrony.
- **Answer:** The discrete-event simulation folks figured this out a few decades ago. We'll just borrow their ideas.

Claim: many practical real-time systems fit this model.

Let's Take Turns!

or Discrete-Event Simulation

- Suppose you have participants p_0, p_1, ..., p_n.

Let's Take Turns!

or Discrete-Event Simulation

- Suppose you have participants p_0, p_1, ..., p_n.
- Suppose the present time is time t.

Let's Take Turns!

or Discrete-Event Simulation

- Suppose you have participants p_0, p_1, ..., p_n.
- Suppose the present time is time t.
- Each p_i is scheduled to take some action in the future, when its timeout is reached.
  - At time $t + n$ (synchronous).
  - Within $(t + n, t + n + \delta)$ (partially-synchronous).

|galois|

Let's Take Turns!

or Discrete-Event Simulation

- Suppose you have participants p_0, p_1, ..., p_n.
- Suppose the present time is time t.
- Each p_i is scheduled to take some action in the future, when its timeout is reached.
  - At time $t + n$ (synchronous).
  - Within $(t + n, t + n + \delta)$ (partially-synchronous).
- The current time t "leap frogs" to the least-valued scheduled action.

Let's Take Turns!

or Discrete-Event Simulation

- Suppose you have participants p_0, p_1, $\ldots$, p_n.
- Suppose the present time is time t.
- Each p_i is scheduled to take some action in the future, when its timeout is reached.
  - At time $t + n$ (synchronous).
  - Within $(t + n, t + n + \delta)$ (partially-synchronous).
- The current time t "leap frogs" to the least-valued scheduled action.

  If two participants have overlapping timeouts, one is arbitrarily chosen.

Let's Take Turns!

or Discrete-Event Simulation

- Suppose you have participants p_0, p_1, ..., p_n.
- Suppose the present time is time t.
- Each p_i is scheduled to take some action in the future, when its timeout is reached.
  - At time $t + n$ (synchronous).
  - Within $(t + n, t + n + \delta)$ (partially-synchronous).
- The current time t "leap frogs" to the least-valued scheduled action.

  If two participants have overlapping timeouts, one is arbitrarily chosen.
- p_i takes its action, and updates its timeout.

Example

$$t = 0$$

p_0	p_1
0	0 from [0, 1]

Example

$$t = 0$$

p_0	p_1
0	0 from [0, 1] $\Longleftarrow$

galois

Example

$$t = 0$$

p_0	p_1	
0	0 from [0, 1]	$\Longleftarrow$
	7	

Example

$$t = 0$$

	p_0	p_1
$\Longrightarrow$	0	0 from [0, 1]
		7

Example

$$t = 0$$

	p_0	p_1
$\Longrightarrow$	0	0 from [0, 1]
		7
	9.89 from (8.5, 10)	

|galois|

Example

$$t = 7$$

p_0	p_1
0	0 from [0, 1]
	7
9.89 from (8.5, 10)	

galois

Example

$$t = 7$$

p_0	p_1	
0	0 from [0, 1]	
	7	$\Longleftarrow$
9.89 from (8.5, 10)		

Example

$$t = 7$$

p_0	p_1
0	0 from [0, 1]
	7 $\Longleftarrow$
9.89 from (8.5, 10)	
	9

Example

$$t = 9$$

p_0	p_1
0	0 from [0, 1]
	7
9.89 from (8.5, 10)	
	9

Example

$$t = 9$$

p_0	p_1
0	0 from [0, 1]
	7
9.89 from (8.5, 10)	
	9 $\Longleftarrow$

|galois|

Example

$$t = 9$$

p_0	p_1	
0	0 from [0, 1]	
	7	
9.89 from (8.5, 10)		
	9	$\Longleftarrow$
	10	

galois

Example

$$t = 9.89$$

p_0	p_1
0	0 from [0, 1]
	7
9.89 from (8.5, 10)	
	9
	10

Example

$$t = 9.89$$

p_0	p_1
0	0 from [0, 1]
	7
$\Longrightarrow$ 9.89 from (8.5, 10)	
	9
	10

Example

$$t = 9.89$$

	p_0	p_1
	0	0 from [0, 1]
		7
$\Longrightarrow$	9.89 from (8.5, 10)	
		9
		10
	(10, 12)	

Example

$$t = 10$$

p_0	p_1
0	0 from [0, 1]
	7
9.89 from (8.5, 10)	
	9
	10
(10, 12)	

galois

Example

$$t = 10$$

p_0	p_1	
0	0 from [0, 1]	
	7	
9.89 from (8.5, 10)		
	9	
	10	$\Longleftarrow$
(10, 12)		

(3) QuickCheck

A Tale of Two QuickCheck Uses

In One Slide

- **# 1 Testing:** we'll feed the model QC-generated real-time parameters satisfying the constraints.

  About 100,000 tests-runs per minute on a MacBook.
- **# 2 Probability Calculating**: QC for use in stochastic testing.
- For both, we use monadic QuickCheck, since the model itself is within the `IO` monad. (A small patch is needed to QC.)
- And we use the super-fast `System.Random.Mersenne` for generating timeouts.
- But with no optimizations, testing is surprisingly fast!

galois

Conclusions

- Emulating real-time is real easy in a pure, lazy language.
- Generating real-time parameters is quick with QuickCheck.
- And QuickCheck can be used for probabilistic reliability analysis.
- Google: `biphase quickcheck` to get the code & QuickCheck patch.

Conclusions

- Emulating real-time is real easy in a pure, lazy language.
- Generating real-time parameters is quick with QuickCheck.
- And QuickCheck can be used for probabilistic reliability analysis.
- Google: `biphase quickcheck` to get the code & QuickCheck patch.

Shameless plug: I'm looking for a summer student (undergrad or Ph.D.) in 2010 and/or 2011 who'd like to do some hacking & research on a NASA-sponsored project...

leepike@galois.com

|galois|